REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
29-09-2012	Final Report	1-Oct-2011 - 30-Jun-2012

4. TITLE AND SUBTITLE

Detection of Low-order Curves in Images using Biologically-plausible Hardware

5a. CONTRACT NUMBER
W911NF-11-1-0271

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
611102

6. AUTHORS
Wesley E. Snyder

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES
North Carolina State University
Research Administration
NC State University
Raleigh, NC 27695 -7514

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
60356-NS-II.1

12. DISTRIBUTION AVAILIBILITY STATEMENT
Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT
Research was conducted to explore the possibility that within the human visual cortex there lies specialized networks which function in a manner similar to that of the Hough transform. Simulation software was written and tested on images. The tests were compared by results reported by psychologists testing human performance, and found to produce consistent results. The conclusion is made that there is no reason to rule out the possibility of Hough-like circuits in the brain which detect straight lines in the peripheral visual field.

15. SUBJECT TERMS
Neural modeling, Hough, straight line detection

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Wesley Snyder
UU	UU	UU	UU		19b. TELEPHONE NUMBER 919-515-5114

Report Title

Detection of Low-order Curves in Images using Biologically-plausible Hardware

ABSTRACT

Research was conducted to explore the possibility that within the human visual cortex there lies specialized networks which function in a manner similar to that of the Hough transform. Simulation software was written and tested on images. The tests were compared by results reported by psychologists testing human performance, and found to produce consistent results. The conclusion is made that there is no reason to rule out the possibility of Hough-like circuits in the brain which detect straight lines in the peripheral visual field.

Enter List of papers submitted or published that acknowledge ARO support from the start of the project to the date of this printing. List the papers, including journal references, in the following categories:

(a) Papers published in peer-reviewed journals (N/A for none)

Received Paper

TOTAL:

Number of Papers published in peer-reviewed journals:

(b) Papers published in non-peer-reviewed journals (N/A for none)

Received Paper

TOTAL:

Number of Papers published in non peer-reviewed journals:

(c) Presentations

Number of Presentations: 0.00

Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

Received Paper

TOTAL:

Number of Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

Peer-Reviewed Conference Proceeding publications (other than abstracts):

Received Paper

TOTAL:

Number of Peer-Reviewed Conference Proceeding publications (other than abstracts):

(d) Manuscripts

Received	Paper

TOTAL:

Number of Manuscripts:

Books

Received	Paper

TOTAL:

Patents Submitted

Patents Awarded

Awards

Graduate Students

NAME	PERCENT_SUPPORTED	Discipline
Theju Jacob	0.47	
FTE Equivalent:	**0.47**	
Total Number:	**1**	

Names of Post Doctorates

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Names of Faculty Supported

NAME	PERCENT_SUPPORTED	National Academy Member
Wesley Snyder	0.20	
FTE Equivalent:	**0.20**	
Total Number:	**1**	

Names of Under Graduate students supported

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Student Metrics

This section only applies to graduating undergraduates supported by this agreement in this reporting period

The number of undergraduates funded by this agreement who graduated during this period:	0.00
The number of undergraduates funded by this agreement who graduated during this period with a degree in science, mathematics, engineering, or technology fields:.....	0.00
The number of undergraduates funded by your agreement who graduated during this period and will continue to pursue a graduate or Ph.D. degree in science, mathematics, engineering, or technology fields:.....	0.00
Number of graduating undergraduates who achieved a 3.5 GPA to 4.0 (4.0 max scale):	0.00
Number of graduating undergraduates funded by a DoD funded Center of Excellence grant for Education, Research and Engineering:	0.00
The number of undergraduates funded by your agreement who graduated during this period and intend to work for the Department of Defense	0.00
The number of undergraduates funded by your agreement who graduated during this period and will receive scholarships or fellowships for further studies in science, mathematics, engineering or technology fields:	0.00

Names of Personnel receiving masters degrees

NAME

Total Number:

Names of personnel receiving PHDs

NAME

Total Number:

Names of other research staff

NAME PERCENT_SUPPORTED

FTE Equivalent:

Total Number:

Sub Contractors (DD882)

Inventions (DD882)

Scientific Progress

See Attachment.

Technology Transfer

Detection of Low-order Curves in Images using Biologically-plausible Hardware

FINAL REPORT
Submitted to the Army Research Office
by
North Carolina State University

Wesley Snyder
Principal Investigator

Covering the period 1 October, 2011 - 1 July 2012

Contents

Chapter 1

Project Description

The work is motivated by the following experience, related to the PI:

> I was looking over a large body of water with land on the other side. The sky was cloudless and blue. Suddenly, I was aware of a flash of a straight line in the upper right of my visual field. When I shifted my gaze to that point, I realized a bird had momentarily flown in such a way as to create a straight line between a water tower and a tree.

This simple experience illustrates several aspects of the perception of straight lines. First, collinearity can be detected in non-foveal areas of the visual field; the observer was not looking at the bird until after his attention was shifted. Second, since directed attention was not on the area of the visual field containing the line, this suggests an operation performed by "hardware" in the visual cortex. Third, this is an unusual phenomenon, it was attention-diverting because the background was so extremely uncluttered, arguing for a system that is sensitive to background clutter.

In later sections of this report, we will refer back to this experience.

In this work, we have sought a biologically-plausible computing architecture which can identify simple features such as straight lines in images. We sought an architecture which will detect these features over very wide expanses of the visual field, not restricted to the fovea or a local receptive field. We sought answers to the following questions:

- whether mammals have "hardware" for straight line detection,
- what the nature of such hardware is, given that only low precision calculations are available,
- how to detect and quantify such functions in the visual cortex, and
- how to build an electronic real-time version.

We have concluded that a computing architecture based on straightforward image analysis operations provides Straight Line Detection (SLD) consistent with human behavior.

Chapter 2

Objectives

In chapter 4.2 we will present a neural architecture which is reasonable to describe the high-speed SLD experience described above. We have simulated this architecture, and tested its performance. We will then describe future work performing comparisons with carefully conducted human behavior experiments to confirm that our proposed and simulated architecture is a reasonable explanation for how human detect straight lines.

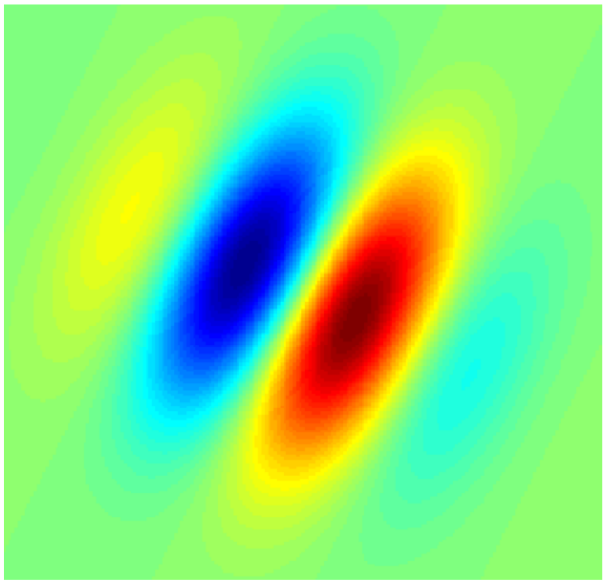


Figure 2.1: A Gabor filter combines derivative of a Gaussian with sinusoidal modulation. The sensitivity of the edge detection neurons is consistent with a Gabor filter. The filter shown in this figure produces an estimate of the first derivative of a brightness edge.

Chapter 3

Technical Background

Since the work of Hubel and Wiesel, (now collected into a book [38]), it has been possible to represent the lower levels of the biological vision system by a convolution with a collection of edge-sensitive kernels. For example, we know by experiment that corresponding to every point in the retina, there are cells which are sensitive to edges/lines with varying orientation.

The output of these cells may be simulated by convolution with an oriented kernel such as a Gabor filter. These feature detectors are known as *Simple cells at layer 1* or just *S1* cells.

3.1 The Standard Model

The *standard model* for vision has been developed in a variety of versions over a number of years by Tomaso Poggio and colleagues. This particular model for vision has always been very careful to avoid using any architectural components that could not be found in a biological brain, and to substantiate any models with actual neural recordings in so far as possible. We follow the same philosophy in this work.

A thorough explanation of the Standard Model is provided in [57]. The model begins with area V1 of the visual cortex. From many neurophysiological experiments, V1 is known to be mapped in a retinotropic way (neurons close to each other in V1 correspond to areas in the visual field which are likewise close to each other). The details of processing done by ganglion cells in the retina, the optic nerve, and the lateral geniculate nucleus (LGN) are lumped into the observation that there are cells in V1 which are sensitive to edges in the image. Different cells are sensitive to edges with different orientations. There seem to be about 4 preferred orientations, horizontal, vertical, and the two 45-degree slanted edges. The output of these cells is consistent with convolution of the image with a Gabor filter (See Figure 2.1)with that specific orientation. The convolution operation may be implemented in simulation by

$$y = g\left(\frac{\sum_{j=1}^{n} w_j x_j^p}{k + \left(\sum_{j=1}^{n} x_j^q\right)^{\tau}}\right) , \tag{3.1}$$

where the w_j are the weights corresponding to the filter, the x_j are the inputs from the LGN, and g is the sigmoid function commonly associated with neural response functions. Refer to [57] for details.

The outputs of the *S1* cells are inputs to *C1* cells, which have larger receptive fields, and which perform a maximum operation over a local neighborhood. By selecting the output from the strongest

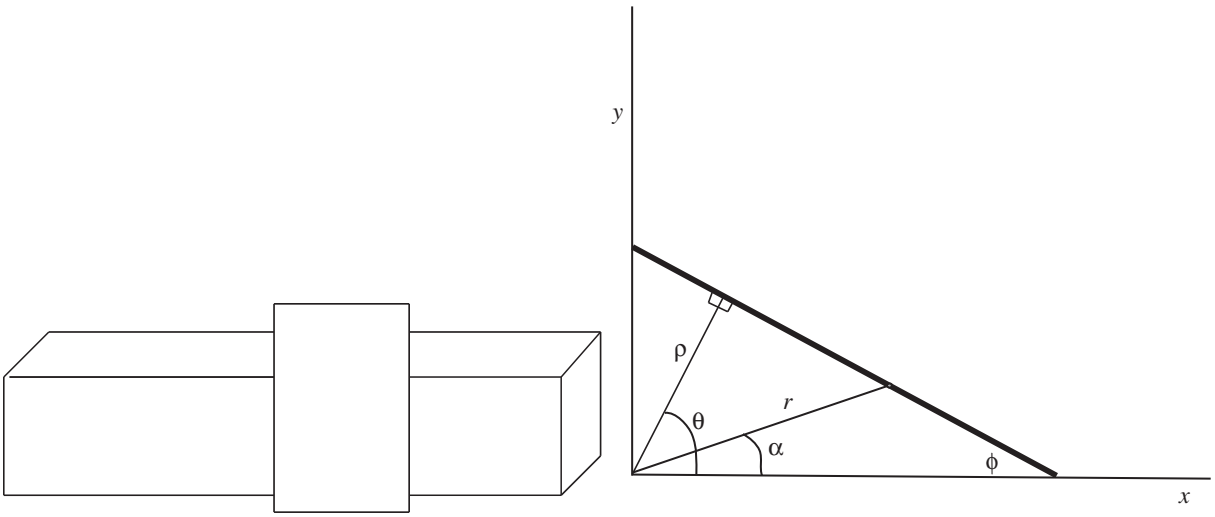


Figure 3.1: LEFT When a straight or gently curving line is occluded, the human visual system infers the existence of the occluded portions of the line. This occurs even when the space between the collinear segments is significantly larger than the observation area of the fovea. RIGHT:A line in the image is parameterized by its distance from the origin (ρ), and the angle it makes with the x axis (θ). Any point on that line may be described using polar coordinates r and α.

S1 cell, and effectively suppressing the others, a small degree of translational invariance is provided, coupled with a more accurate positioning of actual edges. This is similar to the function of the Canny[17] edge operator, which uses nonmaximum suppression to locate the maximum of local edge detector responses. Biologically-plausible implementations of the maximum operation may be described by the "softmax" operation [57] satisfying the equation

$$y = g\left(\frac{\sum_{j=1}^{n} x_j^{q+1}}{k + \sum_{j=1}^{n} x_j^{q}}\right) . \tag{3.2}$$

The maximum operation is also computable using a locally-connected network of neurons using inhibition to implement winner-take-all.

Thus, after the first *S1/C1* pair, edges with specific orientations are identified with considerable precision. This will become the input to our SLD network. In [57], additional levels of *S*- and *C*-like cells are proposed at higher levels of the brain, with increasing size and generality, and shown to provide a feedforward network that can explain foveal shape recognition. However, the *S1-C1* level is all that is required for the SLD operation described in this work.

3.2 The log-polar map

In 1977, Eric Schwartz [55] published his seminal paper illustrating the "log-polar transform". This transform, denoted $L : \mathbf{C} \rightarrow \mathbf{C}$ describes the image plane by an isomorphism to the complex plane, so that a point with coordinates (x, y) in the image is represented by the single complex variable z. Similarly, a point in the cortical plane is represented by $w \in \mathbf{C}$, and the log polar mapping is

$$w = \ln(z) \tag{3.3}$$

This representation is shown to be consistent with a substantial body of neurophysiological literature, and several advantages result from this representation. For example, scale change in the image is converted to a simple translation on the cortex.

The complex number w in magnitude, direction form is denoted $w = (\rho, \theta)$, where

$$\rho = \log\sqrt{x^2 + y^2}$$
$$\theta = \arctan \frac{y}{x}$$

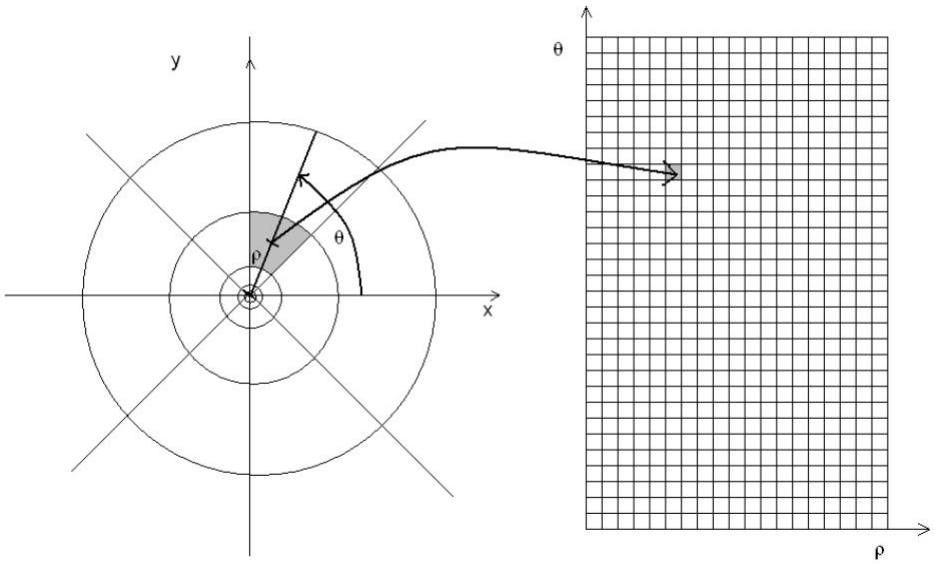


Figure 3.2: Log Polar Transformation

ρ is hence the logarithm of the distance to a given point from the origin and θ is the angle between the axis and the line joining the given point and the origin. In image processing applications, the mapping aids in data reduction by reducing the resolution at image boundaries. The transform is also widely used in applications like image registration, tracking and target recognition.

In Appendix B, the transform is explained in more detail. This analysis was provided by the graduate student supported on this grant, but independent of the grant itself.

Chapter 4

An Architecture for Non-local Vision

A line, or collection of nearly collinear line segments, in an image may be detected by the well-known Hough transform (HT) [36].

The HT is a type of *accumulator method*, and since the original paper, many variations and extensions have been made [7, 19], which follow the Hough philosophy of using accumulation to determine consistency. Some of these methods are based on neural models [12, 13, 8, 9, 10, 11]. In this presentation, we discuss only the original straight-line version.

Accumulator methods allow consistent but spatially diverse image segments to "vote" for consistent interpretations. Because the voting is an addition, zero mean noise is averaged out. The HT methods have been shown [33, 16] to provide highly robust ways to combine local measurements to make global decisions.

1. The method collects input from spatially diverse features in the image and increments a single point/neighborhood in the transform image[1]. Thus, non-connected but almost-collinear features in the image all enhance the same point in the transform space. This produces a means for dealing simply with occlusion. Figure 3.1 shows an example in which the human extrapolates, and infers the presence of straight lines.

2. The accumulation process is simply additions. The method is therefore robust to most additive noise, since positive and negative noise cancels out.

3. Features which are identical (e.g. straight lines) but located in different points in the image transform to different points in the transform space.

We have observed that the advantages of accumulator-based methods are quite significant, and in this work we insisted on finding methods to use them in a biologically-plausible network, though there may be alternatives. For example, Basak [8] develops a method for finding straight lines in images, however, no accumulators are used (although the title of the paper includes the words *Hough Transform*). Instead, a search is made for a collection of weight vectors, each vector representing a straight line. Neural nets methods (weight vector estimation) are used in an iterative algorithm which requires that the maximum allowable number of vectors be known a-priori. In contrast, in this work, we made extensive use of accumulators.

Here, we consider only straight lines, although we have done considerable research in the use of accumulator-based techniques in detecting general shapes[41, 42]. In Figure 3.1(RIGHT) is

[1]Since the transform is implemented using a 2-D array, we will use the word *image*.

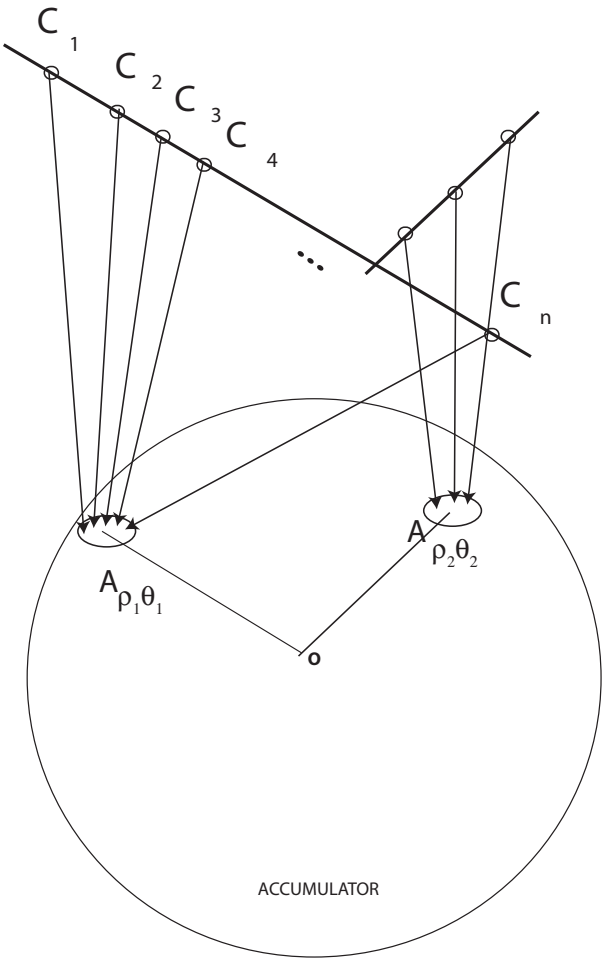


Figure 4.1: Each edge-sensitive cell along the line (ρ, θ) extends a single connection to the the accumulator cell characterizing the edge. In this figure, two edges are illustrated, with orientations of θ_1 and θ_2. The accumulator itself is organized in polar coordinates, so the angle from the center of the accumulator (**O**) is parallel to the line which it represents.

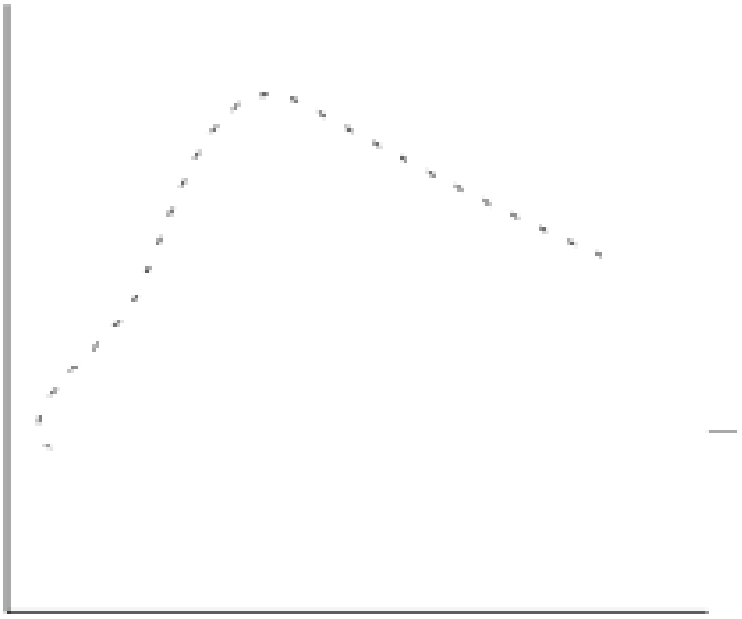


Figure 4.2: An image which is the output from an edge detector. The human can immediately discern that this boundary consists of two straight segments.

illustrated a straight line parameterized by its distance from the origin and the angle, ϕ, it makes with the horizontal axis.

4.1 Background: the Hough Transform

Suppose one is tasked with the problem of finding the straight lines in the image shown in Figure 4.2. If only one straight line were present in the image, we could use straight line fitting to determine the parameters of the curve.

We could represent lines by equations of the form

$$y = ax + b \tag{4.1}$$

But we have two line segments here. If we could segment this first, then we could fit each segment separately – yes, this is a segmentation problem – but we are segmenting a boundary into boundary segments rather than segmenting an image into regions. In this section, we will learn how to do this.

First, let us prove an illustrative theorem.

Definition: given a point in a d-space, and a parametrized expression defining a curve in that space, the parametric transform of that point is the curve which results from treating the point as a constant and the parameters as variables. For example, Eq. 4.1 produces the parametric transform

$$b = y - xa \tag{4.2}$$

which is itself a straight line in the 2-space $< a, b >$. Given the point $x = 3, y = 5$, then the parametric transform is $b = 5 - 3a$.
Theorem: If n points in a 2-space are collinear, all the parametric transforms corresponding to those points, using the form $b = y - xa$ intersect at a common point in the space $< a, b >$.
Proof: Suppose n points $\{x_1, y_1), (x_2, y_2), , , (x_n, y_n\}$ all satisfy the same equation

$$y = a_0 x + b_0 \tag{4.3}$$

Consider two of those points, (x_i, y_i) and (x_j, y_j) . The parametric transforms of the points are the curves (which happen to be straight lines)

$$y_i = x_i a + b \tag{4.4}$$
$$y_j = x_j a + b \tag{4.5}$$
$$\tag{4.6}$$

which we rewrite to make clear the fact that a and b are independent variables

$$b = y_i - x_i a \tag{4.7}$$
$$b = y_j - x_j a \tag{4.8}$$

The intersection of those two curves, straight lines in a, b is a single point.
Solving the two equations of Eq. 4.8 simultaneously results in

$$y_i - y_j = (x_i - x_j)a \tag{4.9}$$

and therefore $a = \frac{y_i - y_j}{x_i - x_j}$.
We substitute a into Eq. 4.6 to find b,

$$b = y_i - x_i \frac{y_i - y_j}{x_i - x_j} \tag{4.10}$$

and we have the a and b values where the two curves intersect. However, we also know from Eq. 4.3 that all the xs and ys satisfy the same curve. By performing that substitution into Eq. 4.10, we obtain

$$b = (a_0 x_i + b_0) - x_i \frac{((a_0 x_j - b_0) - (a_0 x_i + b_0))}{x_i - x_j} \tag{4.11}$$

which simplifies to

$$b = (a_0 x_i + b_0) - x_i a_0 = b_0 \tag{4.12}$$

Similarly,

$$a = \frac{y_i - y_j}{x_i - x_j} = \frac{(a_0 x_i + b_0) - (a_0 x_j + b_0)}{x_i - x_j} = a_0 \tag{4.13}$$

Thus, for any two points along the straight line parameterized by a_0 and b_0, their parametric transforms intersect at the point $a = a_0$ and $b = b_0$. Since the transforms of any two points transforms intersect at that one point, all such transforms intersect at that common point. QED.
Review of concept: Each POINT in the image produces a CURVE (possibly straight) in the parameter space. If the points all lie on a straight line in the image, the corresponding curves will intersect at a common point in parameter space.

The Problem with Vertical Lines In order to deal with vertical lines, in which the slope, a, is infinite, we choose a different form for the straight line

$$\rho = x \cos\theta + y \sin\theta \tag{4.14}$$

For a given pair of values for ρ and θ, the set of points which satisfy Eq. 4.14 can be shown to be a straight line.

This representation of a straight line has a number of advantages. Unlike the use of the slope, both of these parameters are bounded; ρ can be no larger than the largest diagonal of the image, and θ need be no larger than 2π. A line at any angle may be represented without singularity. The use of this parameterization of a straight line solves one of the problems which confronts us, the possibility of infinite slopes. The other problem is the calculation of intersections.

How to Find Intersections – Accumulator Arrays

It is not feasible to find all intersections of all curves, and to then determine which of those are close together. Instead, we make use of the concept of an accumulator array. To create an accumulator array, we make an image, say 360 columns by 512 rows. We initialize each of the pixels to zero. From now on, we will refer to the pixels of this special image as accumulators. Figure 4.4 illustrates plotting two straight lines through an accumulator array using the following algorithm:

- For each point (x_i, y_i) in the edge image, do
    1. for all values of θ compute ρ.
    2. at the point ρ, θ in the accumulator array, increase the value at that point by an amount proportional to the strength of the edge.

This algorithm results in multiple increments of those accumulators corresponding to intersections being increased more often. Thus the peaks in the accumulator array correspond to multiple intersections, and hence to proper parameter choices.

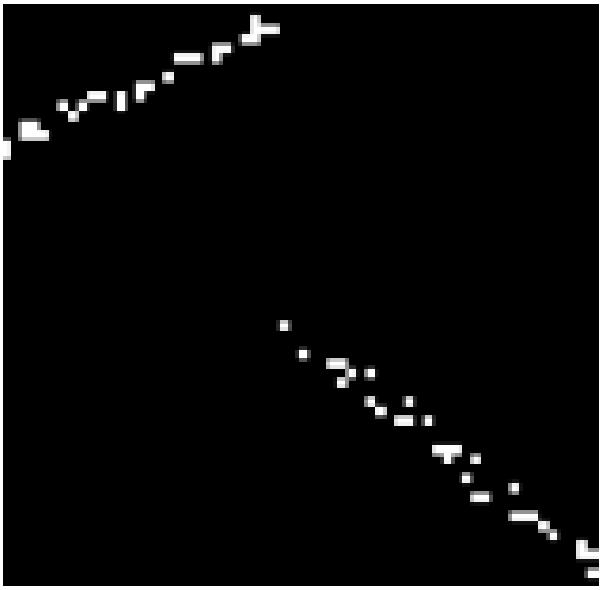


Figure 4.3: Two very noisy lines.

In this work, we do not use the Hough transform as described in this subsection (except for test validation), however, we do make use of accumulators to find consistent contributions to collinear segments.

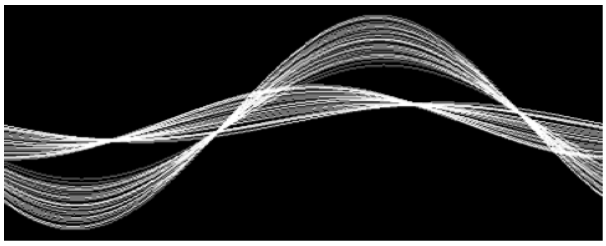


Figure 4.4: The Hough Transform of figure 4.3

4.2 The Architecture

There are two arrays of interest, the image and the accumulator. The image is defined by the output of *C1* cells, and denoted $C1(x, y, \theta)$. The spatial coordinates of the *C1* cell could be Cartesian (x, y) or polar (r, ϕ). In the proposed system, either may be used, but we choose Cartesian simply because it is easier for humans to visualize. We choose to use the output of *C1* cells because those cells reflect the results of two lower-level stages of processing, convolution and softmax. The convolution is performed by *S1* cells as described in section 3.1, and similarly, the *C1* cells provide a degree of invariance to small rotations and translations, also as described in section 3.1. The third parameter of the image function, θ (see figure 3.1) is an encoding of the orientation of the angle passing through the receptive field. We discuss how this encoding is developed elsewhere in this document.

Each cell $A(\rho, \theta), A : \Re \times (0, 2\pi] \rightarrow \Re$ in the accumulator array A may represent a line or an edge in the image. For this initial work, we limit the exploration to detecting lines. The variable θ is the same θ as mentioned in the previous paragraph, and parameterizes the slope of the edge. As in Figure 4.1, ρ is the minimum distance of this edge to the foveola[2].

The accumulation function used here relies on a critical observation: At any point in *C1*, **both the spatial coordinates, and the orientation of the edge are known, therefore the edge is uniquely determined**. For this reason, only one[3] afferent connection is required from a *C1* cell in V1, $C1(x, y, \theta)$, to a cell in the accumulator, $A(\rho, \theta)$, as illustrated in Figure 4.1. The accumulator may be considered as a two dimensional array indexed by θ and ρ. It is straightforward to show that for any point x, y on the line,

$$\rho = (x \cos\theta + y \sin\theta) \tag{4.15}$$

The accumulator cells have as many efferents as there are *C1* cells in the corresponding edge, which could be on the order of a thousand, still a reasonable biological assumption.

Here, we represent a straight line by a finite set of points (r, θ). The set of points constituting line Z is defined by

$$A(\rho, \theta) = \{r, \theta| \quad |\rho - r\cos\theta - r\sin\theta| < \delta\} \tag{4.16}$$

where δ is some "small" positive constant. Thus, the accumulator cell with parameters (ρ, ϕ) has a value equal to

$$A_{\rho,\theta} = \sum_{i}^{n} C1_i \tag{4.17}$$

[2]Although the foveola, the center of the fovea, has a non-infinitesimal area, it is quite small, and the term is used here to denote the center of the foveola, the origin of the coordinate system

[3]One connection is not robust, however, and this is addressed in section 5.2

where i simply denotes a cell on the line ρ, θ. The fact that the orientation returned by edge detectors is the same as the parameter used by the transform potentially provides for the simple interconnection architecture mentioned above.

Chapter 5

Approach

In this section, we describe topics addressed in the initial (STIR) phase of the project, and topics likely to be addressed in the second, follow-on, phase of the project. In this way, the reader may see the entire scope of the effort.

In this, the "Approach" section, we will describe experiments run and also the results of those experiments.

5.1 Estimating Orientations

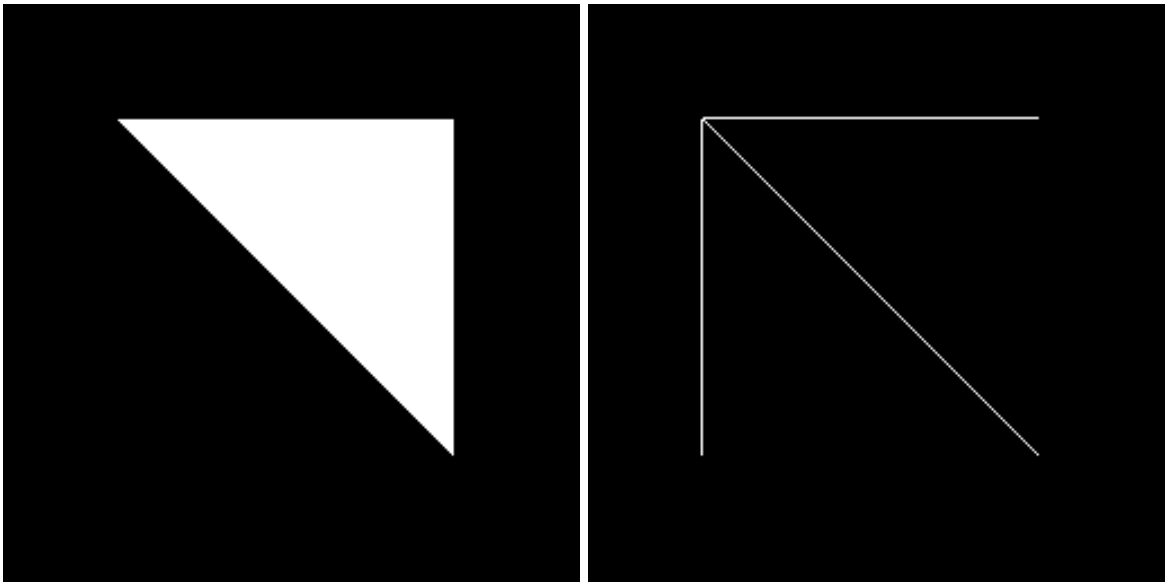
Schoups et al. [51] indicate that there is no particularly preferred orientation in the visual field of monkeys. Using a fixed, small number of Gabor-like filters suggests a reasonable biological implementation, just a few orientation-sensitive cells at every point. Yet, we must ask, if we only make, say, four independent measurements of orientation, and each one is accurate only to, say, one part in thirty (five bits precision), how can eight-bit precision occur, as has been measured in primates[51]? In Appendix A, we show how four measurements of the orientation of a particular edge can be used to produce an accurate estimate of the actual orientation. However, though that appendix demonstrates that a linear machine can determine a single estimate from the four measurements, it does not consider in detail how a biological collection of cells could perform this operation.

In this phase of the research, we have examined the method described in the appendix, as well as a number of other methods to validate how well they work on long and short line segments and how tolerant they are to errors in the original measurements. In this phase, we also determine the best way to use a set of measurements directly to select the cells or cell neighborhood in A to be accumulated. In Figure 5.1, we illustrate the result of a simulation using multiple neurons with only a few bits of precision. By exploiting consistency (as determined by the accumulator), high precision results can be obtained from these low precision calculations.

In the following subsections, we explore a variety of options for estimating the orientation of an edge from a limited number of measurements. We assume n measurements of orientation have been made, and we wish to find a best estimate of the actual orientation. The interpolation is to be accurate over a range of 0 to π which has been quantized into m possible angles. For convenience, we choose m to be 180, so that a change of $\Delta\theta = \pi/180$ corresponds to one degree.

Typically, we might make 6 measurements using the Gabor filters, and interpolate those 6 measurements to an accuracy of one degree.

In the following explanatory sections, we will use one of the simple inputs shown below:



We experimented with a number of ways in increment the accumulator based on a set of directional measurements. These are described below:

5.1.1 Estimating Orientation Using the Traditional Hough Transform (mode 0)

Mode 0 is the classical Hough Transform (HT). At each point, the entire accumulator is incremented. First, the maximum strength of the response is tested, and if it is significant, a plot of the curve $\rho = x\cos\theta + y\sin\theta$ is generated by allowing θ to range over permissible values. We implemented this method only because it provides a reference to a well-known method. Interestingly, we found the Hough Transform to not only be slower, but less accurate than other methods, probably due to the fact that peaks in the transform space tend to not be symmetric. Experimentally, it works well, but other estimators are faster, and the HT does not meet the requirement to be biologically plausible. Figure 5.3 illustrates the accumulator produced by the classical HT, and the image reconstructed.

5.1.2 Estimating Orientation By Interpolation (mode 1)

This mode uses the n measurements of directional derivative magnitude and then linearly interpolates them over a total of 180 1-degree measurements. Figure 5.4 illustrates the results on the test image. Unfortunately, this method seldom works, probably due to the idea that the true angle may be derived from a single linear interpolation is simply false. This method may be compared with mode 5, which uses a parabolic interpolator and works very well.

5.1.3 Estimating Orientation Using the Pseudo-inverse (mode 2)

In Appendix A, we illustrate how a gradient vector may be estimated from 4 noisy estimates. It is straightforward to extend this to a larger number of estimates than 4, and we did so.

A single noisy gradient vector, projected onto all four directions produces four (or n in the general case) noisy measurements. The method then returns the original vector quite precisely. However, if the n measurements are not simply projections (as the Gabor does not return strict projections), the estimated value may be quite imprecise. We ended up unable to use this method for $n > 4$.

5.1.4 Estimating Orientation Using the maximum Gabor Output (mode 3)

At each point, a number, n of filters have been applied. The largest response is chosen as the correct angle. The problem with this approach is illustrated in figure 5.2 which shows a single line which has been sampled by a collection of filters several times. The line subtends an angle of 30 degrees with the x axis. Assuming there are four Gabor filters (0, 45^o, 90^o, 135^o). In each case, the 45^o filter will have the strongest output. However, if no interpolation is done, those four responses will not align. Figure 5.5 illustrates the nearly perfect results on the test image.

5.1.5 Estimating Orientation by Incrementing all the Directions (mode 4)

Because all n directions are incremented, a significant amount of blur allows a successful detection. This mode is attractive because no interpolation is needed at all, except for the natural summing of the measurements. Furthermore, this is the mode originally envisioned in the proposal. Surprisingly, it also has the best performance. Figure 5.6 illustrates the results on the test image.

5.1.6 Estimating Orientation by Fitting a Parabola (mode 5)

First, find the maximum response from the n. Let that angle be denoted θ_2 and the filter output at that point be y_2. Let the angle measurement on the left be θ_1 with output y_1 and similarly the point on the right be θ_3, y_3.

Let the distance (in degrees) be γ, and move the origin to allow θ_2 be zero. A parabola which passes through these 3 points satisfies

$$y_1 = a(-\gamma)^2 - b\gamma + c \tag{5.1}$$
$$y_2 = c \tag{5.2}$$
$$y_3 = a(\gamma)^2 + b\gamma + c \tag{5.3}$$

This system of equations is satisfied by

$$c = y_2 \tag{5.4}$$
$$b = (y_3 - y_1)/2\gamma \tag{5.5}$$
$$a = \frac{y_3 - b\gamma - c}{\gamma^2} \tag{5.6}$$

Then, the location of the strongest response is found by differentiating the parabola and setting the result to zero producing

$$\hat{\theta} = -b/2a \tag{5.7}$$

This approach can only fail in the degenerate case that all 3 y's are the same. In that case, the estimated direction is simply that of θ_2.

Numerous experiments confirm this approach works best. The only problem is the division of equation 5.7 is challenging to motivate in a biological network. Figure 5.7 illustrates the results on the test image;

5.1.7 Estimating Orientation by Linearly Interpolating the Strongest Two Directions (mode 6)

One other approach is to choose only two measurements, the one having the strongest response, and one of its neighbors, and performing a linear interpolation. This approach does not work because

this would require that the interpolation be less than the already-measured maximum. Figure 5.8 illustrates the results on the test image.

5.2 Accumulation

A single connection from the image plane to the accumulator raises the possibility that several nearby points in the accumulator may be each individually incremented, without one particular point emerging as the clear representative. In the traditional Hough transform, this is most frequently dealt with by blurring the accumulator.

In the traditional Hough transform, however, the accumulation of points near a peak becomes problematic because the distribution of points is not isotropic. Figure 4.4 illustrates a traditional Hough Transform with distinct peaks. Even though the peaks are distinct, near the peaks the distribution is strongly non-isotropic, and a large-scale smoothing model will not be appropriate.

We attempted to solve the problem of finding the peak by inventing a new clustering algorithm which we called *BlackHole*. In this approach, each nonzero point in the accumulator represents a particle in free space with a mass equal to its brightness, and moving in response to the net gravitational field of all the points. Gradually, the points all collapse into a single very bright spot which will be the cluster center.

Since this effort was a bit of a diversion from the main emphasis of this project, we allocated only one person-week to it. Unfortunately, the problem turned out to have a number of subtle details, was in general more complex than we had anticipated, and after one person-week, we had to abandon the effort before it yielded a solution. Fortunately, modes 3 - 5 yield tight, dense clusters which are relatively round. We found that simple, small area blurring, followed by local maximum detection was sufficient for modes 3, 4, and 5.

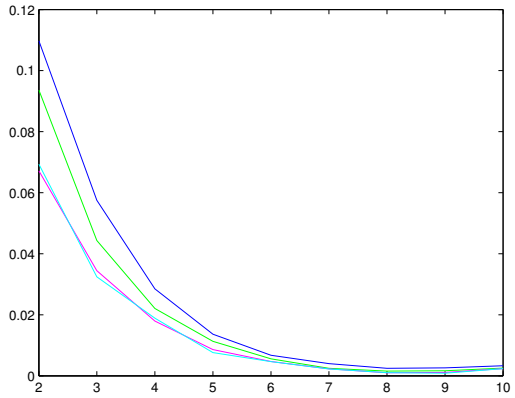


Figure 5.1: Graphs show average precision (measured in radians) in determining the orientation of a straight line, as a function of the number of bits of accuracy used in the estimate. From top to bottom, the curves illustrate the effects of having 2, 3, 5, and 10 points along that line. It is clear that more than ten points do not increase the precision. The parameters are determined by choosing points roughly along the line ("rough" is determined by the bits of precision), and then measuring the ρ, ϕ parameters of the line. These parameters are also computed only to the precision of the bits specified along the horizontal axis. Since the possible range was π, an accuracy of 0.01 at 5 bits is equivalent to one part in 300, or over eight bits of precision.

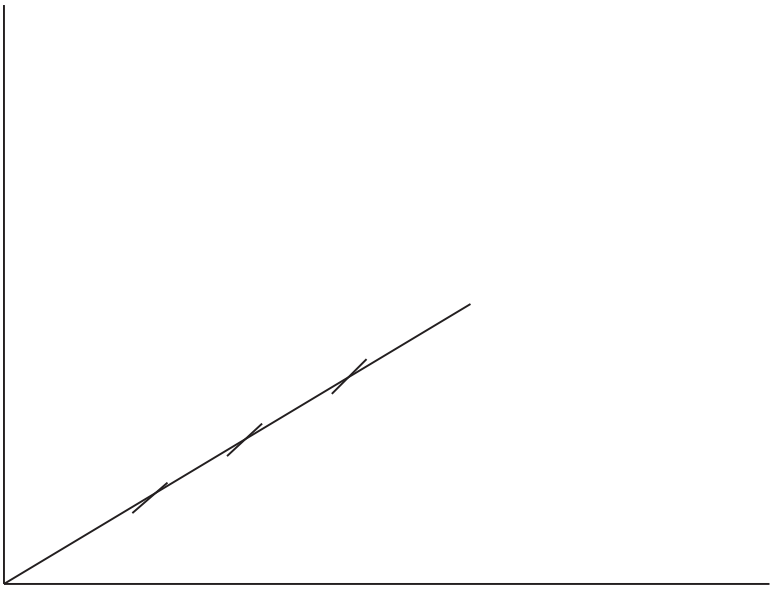


Figure 5.2: If the orientation resolution is too small (an insufficient number of directional filters is used), the sampled points will not be collinear

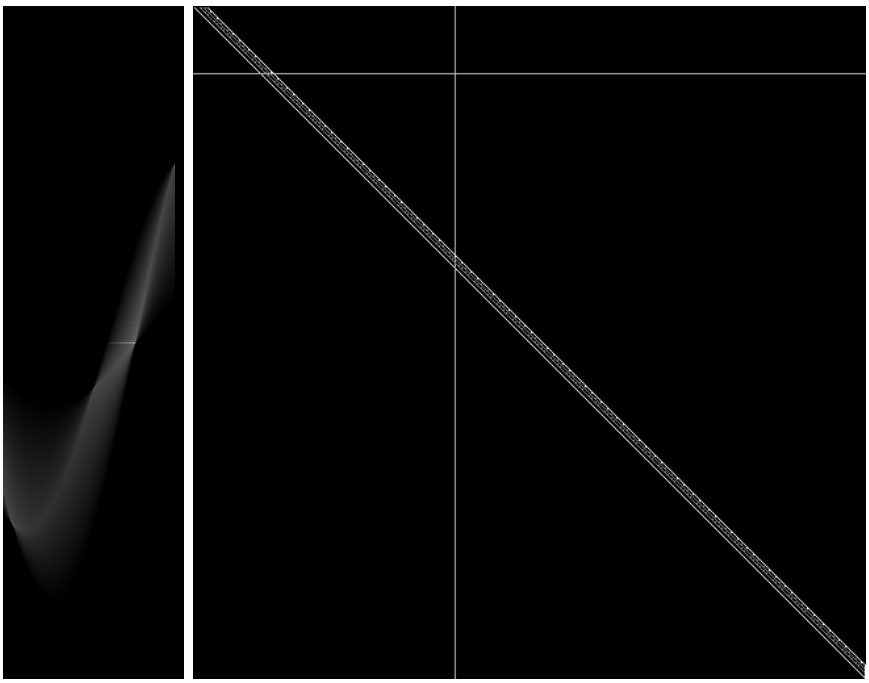


Figure 5.3: (MODE 0) LEFT:Accumulator resulting from finding lines in the test image, using the traditional Hough Transform. RIGHT: The image reconstructed from the accumulator. Note that the output image (512×512) is larger than the input image (256×256), so the reconstructed image shows the edges in the upper left quadrant.

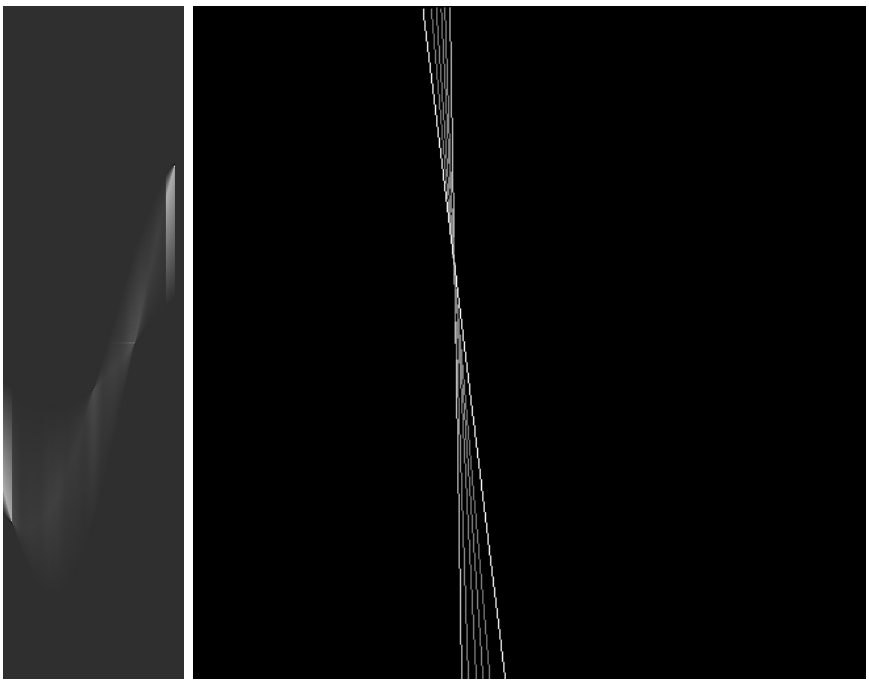


Figure 5.4: (MODE 1) LEFT:Accumulator resulting from finding lines in the test image, using mode 1, which does not work. RIGHT: The image reconstructed from the accumulator.

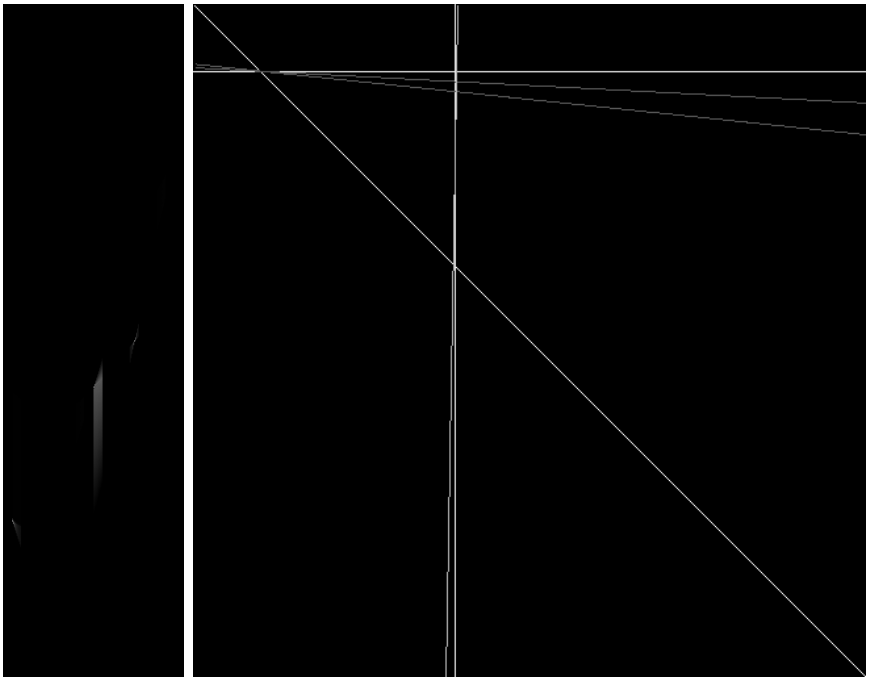


Figure 5.5: (MODE 3) LEFT:Accumulator resulting from finding lines in the test image, using mode 3, which works well. RIGHT: The image reconstructed from the accumulator. Note that the output image (512×512) is larger than the input image (256×256), so the reconstructed image shows the edges in the upper left quadrant.

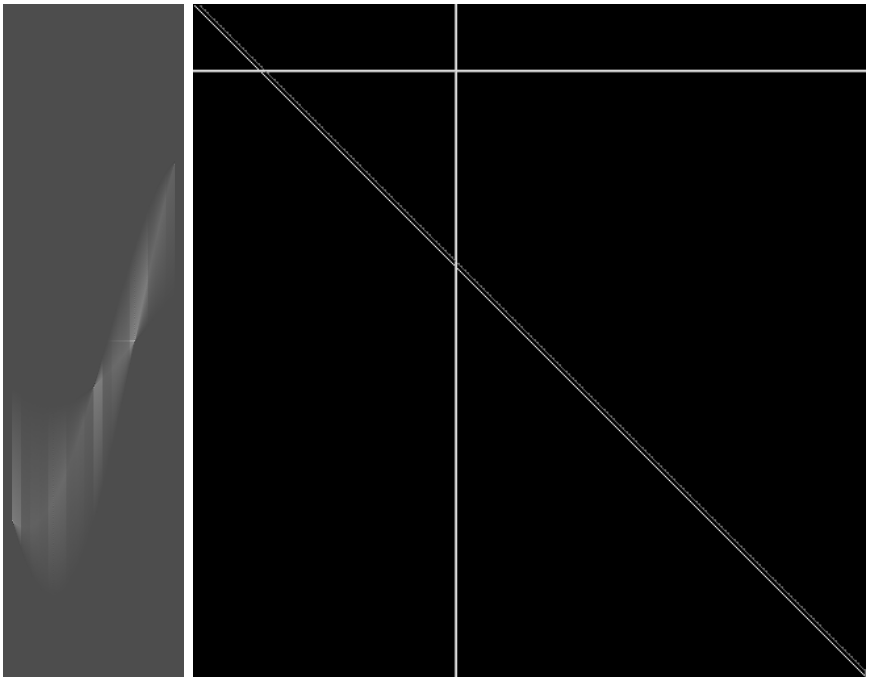


Figure 5.6: (MODE 4) LEFT:Accumulator resulting from finding lines in the test image, using mode 4, which works well. RIGHT: The image reconstructed from the accumulator. Note that the output image (512×512) is larger than the input image (256×256), so the reconstructed image shows the edges in the upper left quadrant.

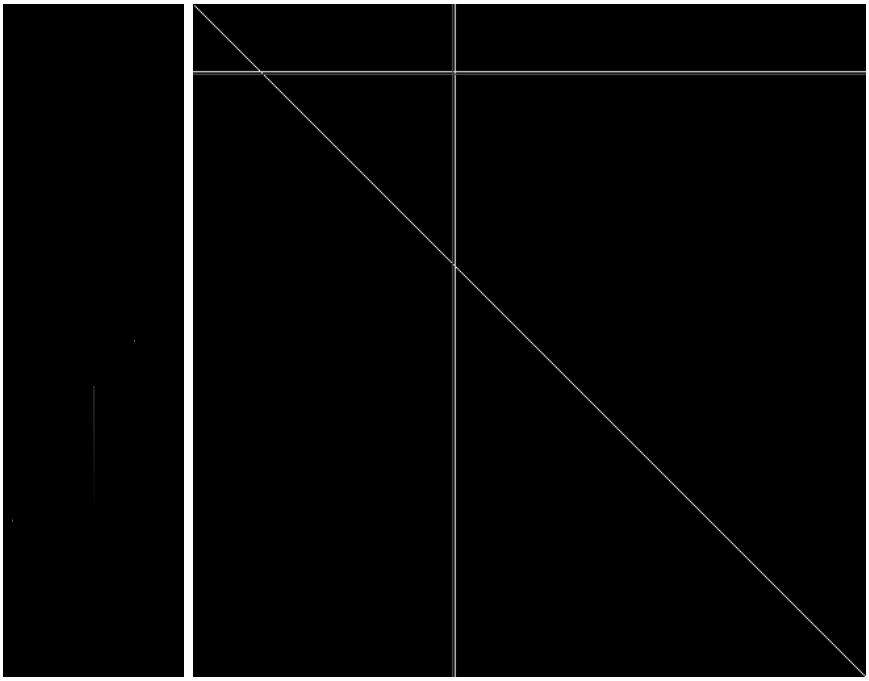


Figure 5.7: (MODE 5) LEFT:Accumulator resulting from finding lines in the test image, using mode 5, which works well. RIGHT: The image reconstructed from the accumulator. Note that the output image (512×512) is larger than the input image (256×256), so the reconstructed image shows the edges in the upper left quadrant.

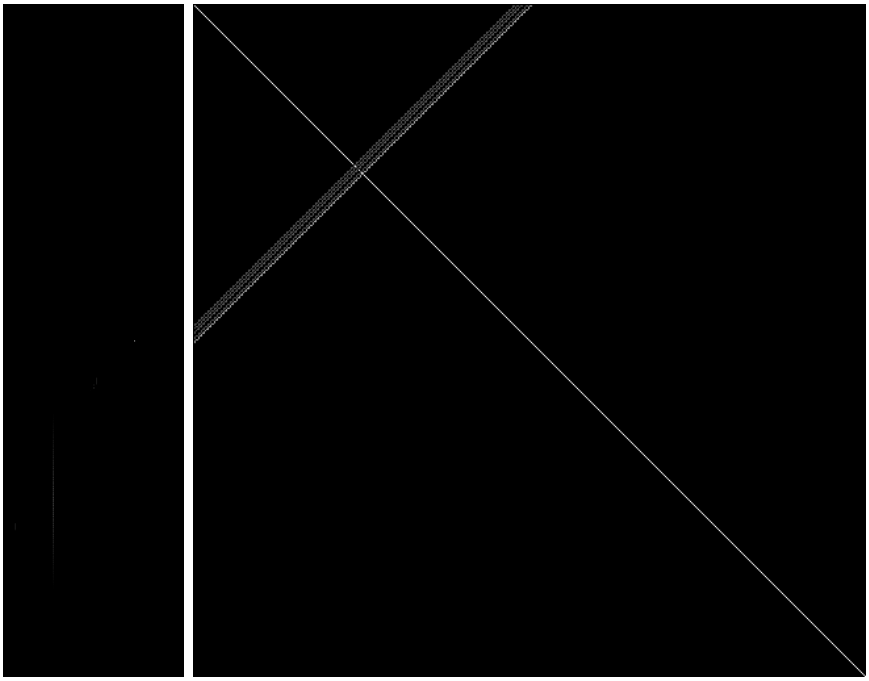


Figure 5.8: (MODE 6) LEFT:Accumulator resulting from finding lines in the test image, using mode 6, which does not work well. RIGHT: The image reconstructed from the accumulator. Note that the output image (512×512) is larger than the input image (256×256), so the reconstructed image shows the edges in the upper left quadrant.

Chapter 6

Results

Figure 6.1 illustrates the type of image we used for testing. Instead of natural images, we used this class of test images because it is possible to carefully control the parameters of interest. The reader should look at the first image and identify the most distinctive straight line in that image. Most human observers choose the straight line at an angle of -45^o to the x-axis in the lower right. That line consists of five microlines. However, there are also two other lines of interest in this image, a horizontal line at the upper left , and another horizontal line about 1/3 of the way down. Although both of those macrolines are only of length 2, nonetheless, they are collinear segments, and we would expect a program which models human behavior to detect them.

This experiment was motivated by previous work in neural modeling [67, 59, 30], using similar test images. Gintautas et al. state "we employ abstract computer-generated shapes consisting of short, smooth contour segments that could either be globally aligned to form wiggly, nearly closed objects (*amoebas*, or else randomly rotated to provide a background of locally indistinguishable contour fragments (*clutter*)." In our case, we seek not amoebas but lines, but the philosophy is identical. By using test images like this, we can control a variety of parameters such as:

length Here, the term *length* refers to the length of the macroline. It is the total number of microlines which make up the macroline. For example, the length of the 45^o macroline referred to above is 5.

Microline orientation A macroline consists of one or more microlines, and each microline has an orientation. The value *orientation* is actually the variance of the orientation of the microlines which make up a single macroline. Performance as a function of variance of microline orientation is discussed in section 6.5.3.

We have tested the peak detection algorithm extensively, and will show performance results later in this section. First, we discuss the nature of the data and the experiments.

6.1 The Experiments

Using modes 3, 4, and 5, we process this image, find the accumulator, and reconstruct the accumulator to produce Figures 6.2 - 6.4. The brightness of the line in the reconstructed image is proportional to the confidence the algorithm has in the detection.

We then repeat this experiment using the second image. Remember, these two images differ only in the small random perturbation applied to the diagonal line. Interestingly, all three methods

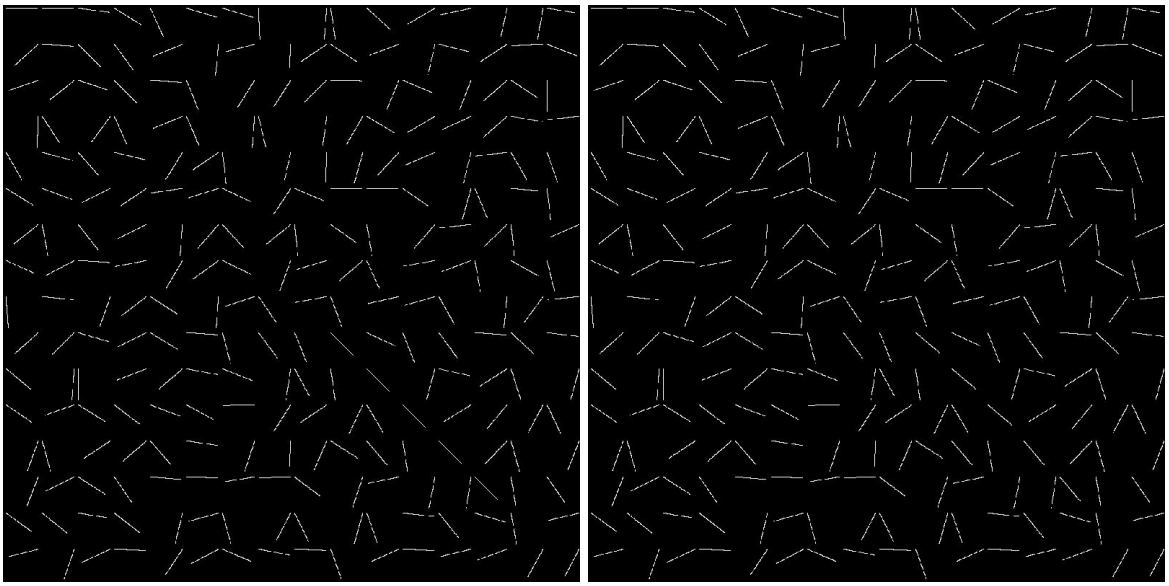


Figure 6.1: Two similar images from the set of test images. The 45^o line in the lower right of the left image is deformed by a 5^o random variation to form the image on the right. Note, on some computer screens, lines may disappear due to the coarse sampling required to show both images on the same page of the report. Also note that aliasing effects occur in these images which were corrected as illustrated in Figure 8.1

fail to identify the diagonal line as having high likelihood. We have run experiments[1] with humans and found a similar result, although humans will sometimes identify the dark region around the diagonal line as a collinear segment. Even small variations away from collinearity caused significant changes in the human's sensitivity to lines.

6.2 Detection as a Function of Length

Figure 6.8 illustrates the probability of correct detection as a function of length of the macroline. This figure is an average over a variety of values for variation in microline orientation and macroline orientation.

6.3 Detection as a Function of Microline Orientation

Figure 6.9 shows the probability as a function of variance in the orientation of the microlines. It is particularly interesting to compare the performance of the method with the performance of a human in this context. For example, note the two images in Figure 6.1 differ only by the random variation of the microlines in the single 45^o line, and that variance is only five degrees. Yet, when the simulation is run on the left image, the 45^o line is perceived as the most distinctive, but when it is run on the right image, the horizontal line near the center of the image replaces the 45^o line as the most distinctive. Several humans that we polled agree with this distinction, suggesting that the model is consistent with human behavior.

[1]The results of these experiments can only be classified as anecdotal, as they were performed without an expert in human cognitive psychology to design them.

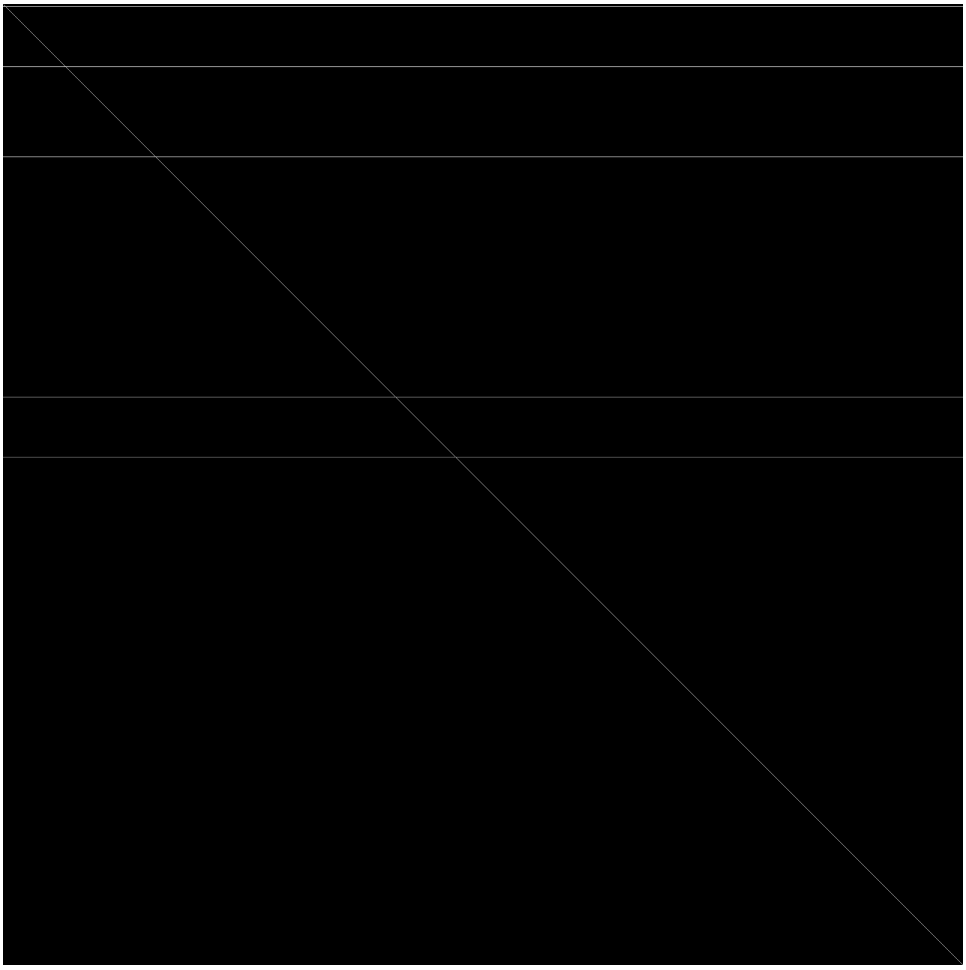


Figure 6.2: The first test image processed with mode 3.

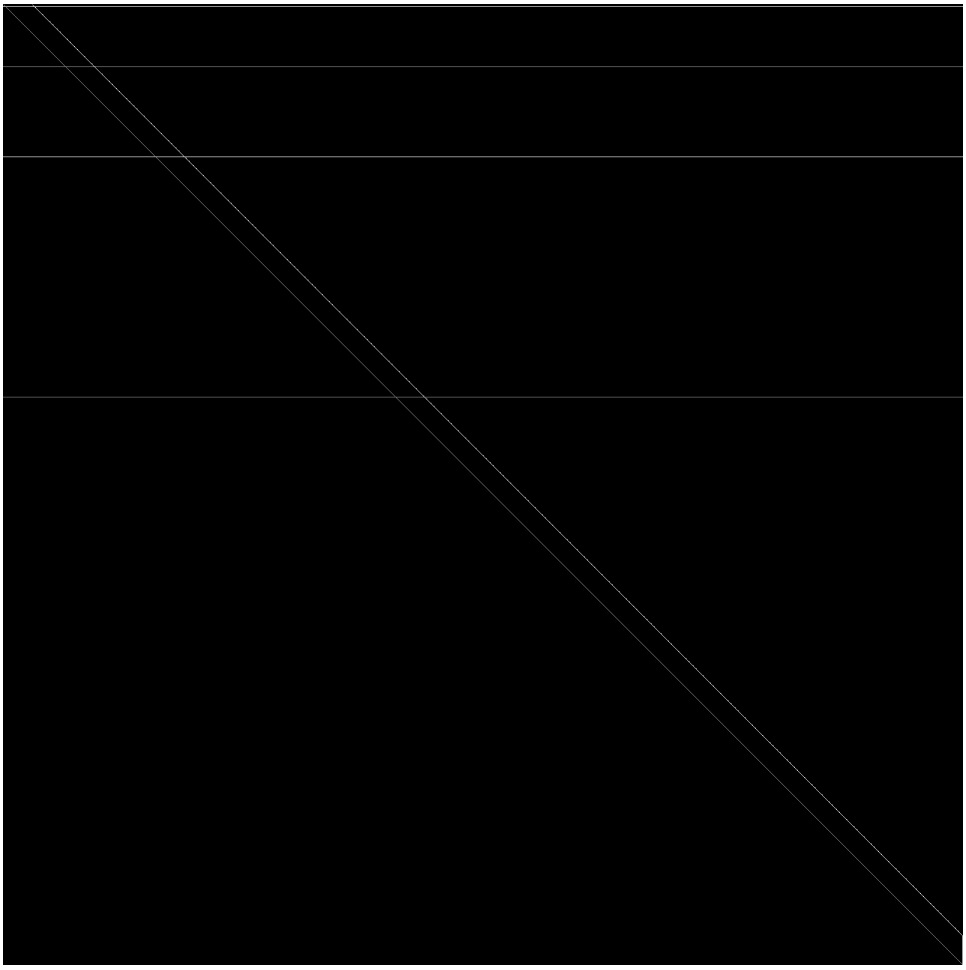


Figure 6.3: The first test image processed with mode 4.

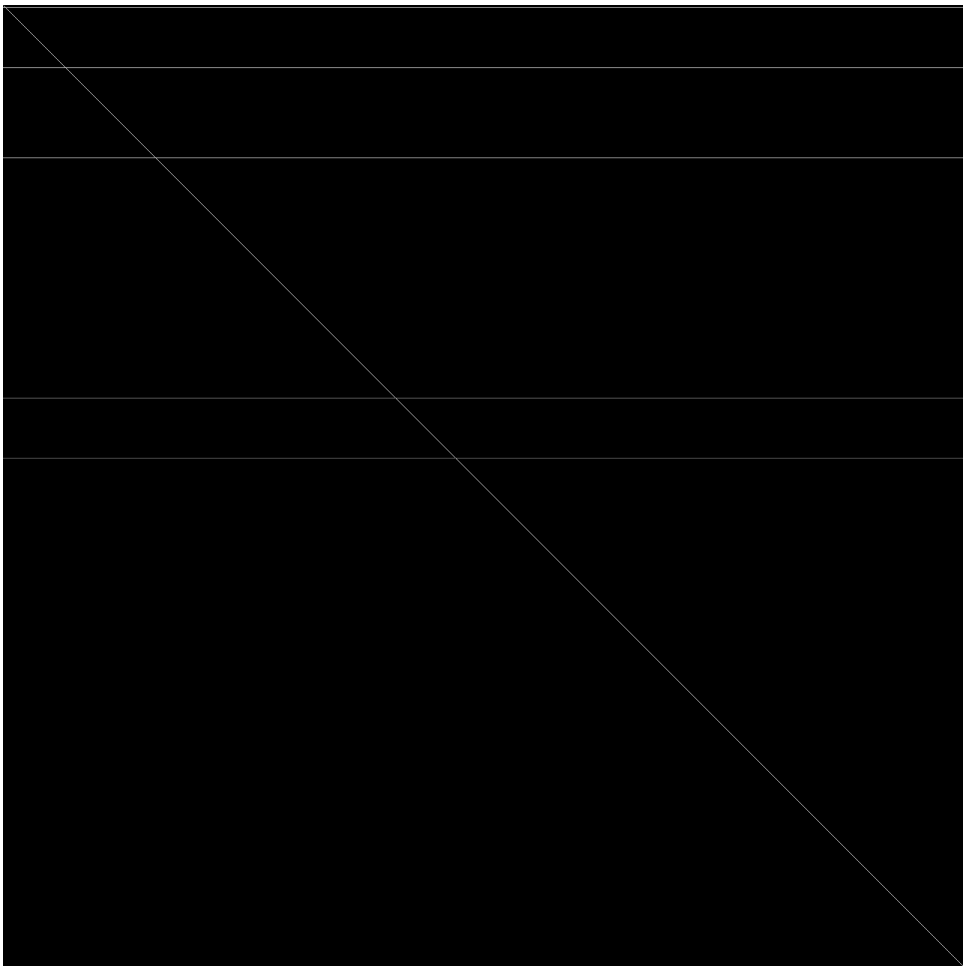


Figure 6.4: The first test image processed with mode 5.

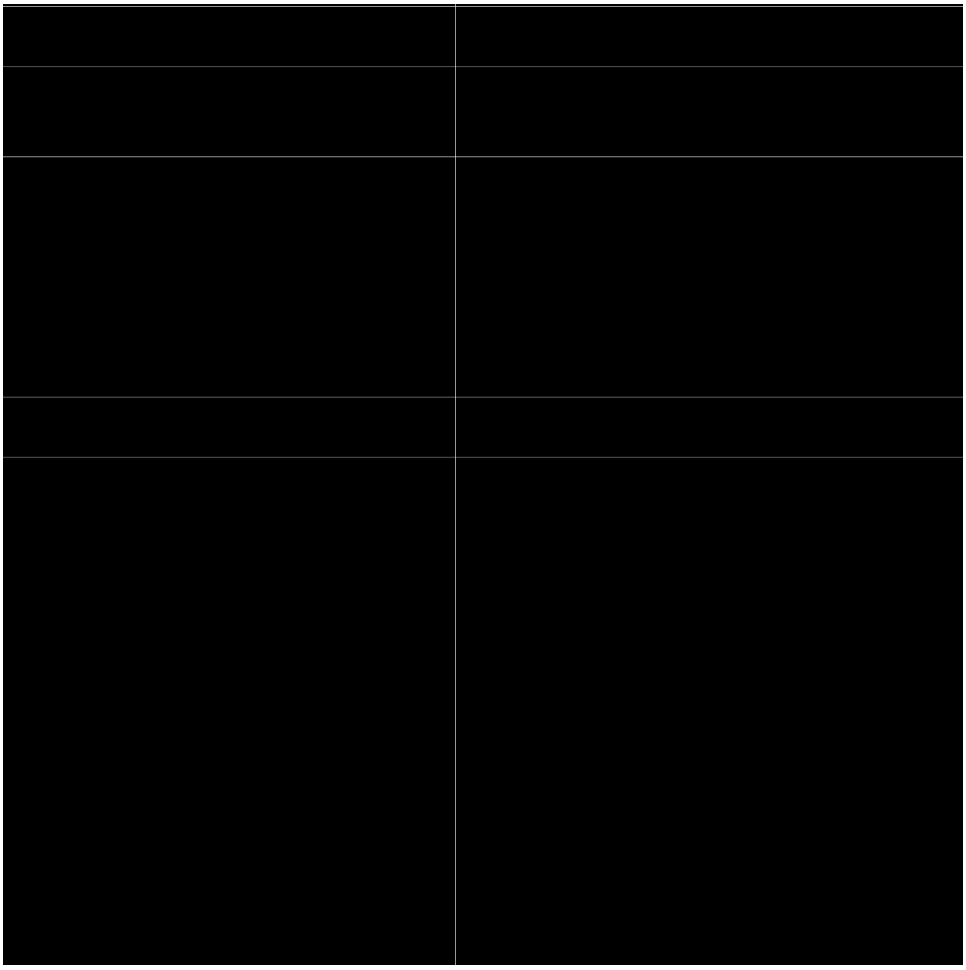


Figure 6.5: The second test image processed with mode 3.

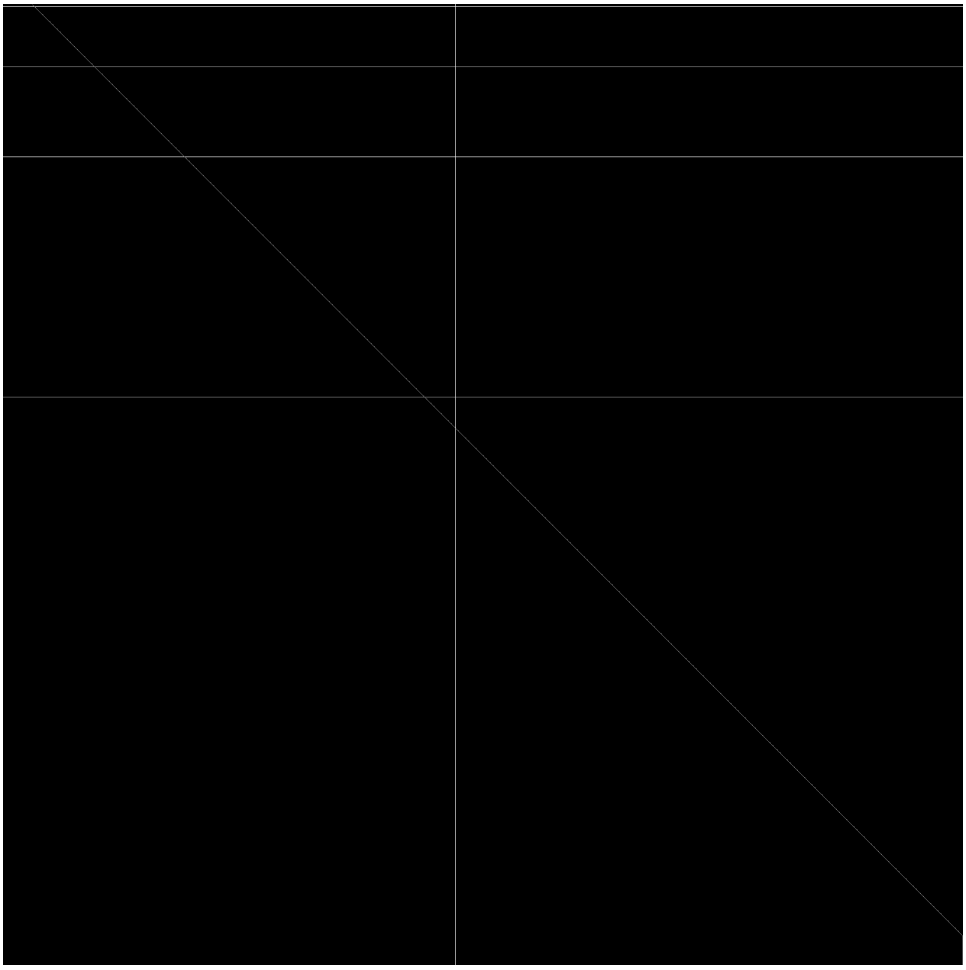


Figure 6.6: The second test image processed with mode 4.

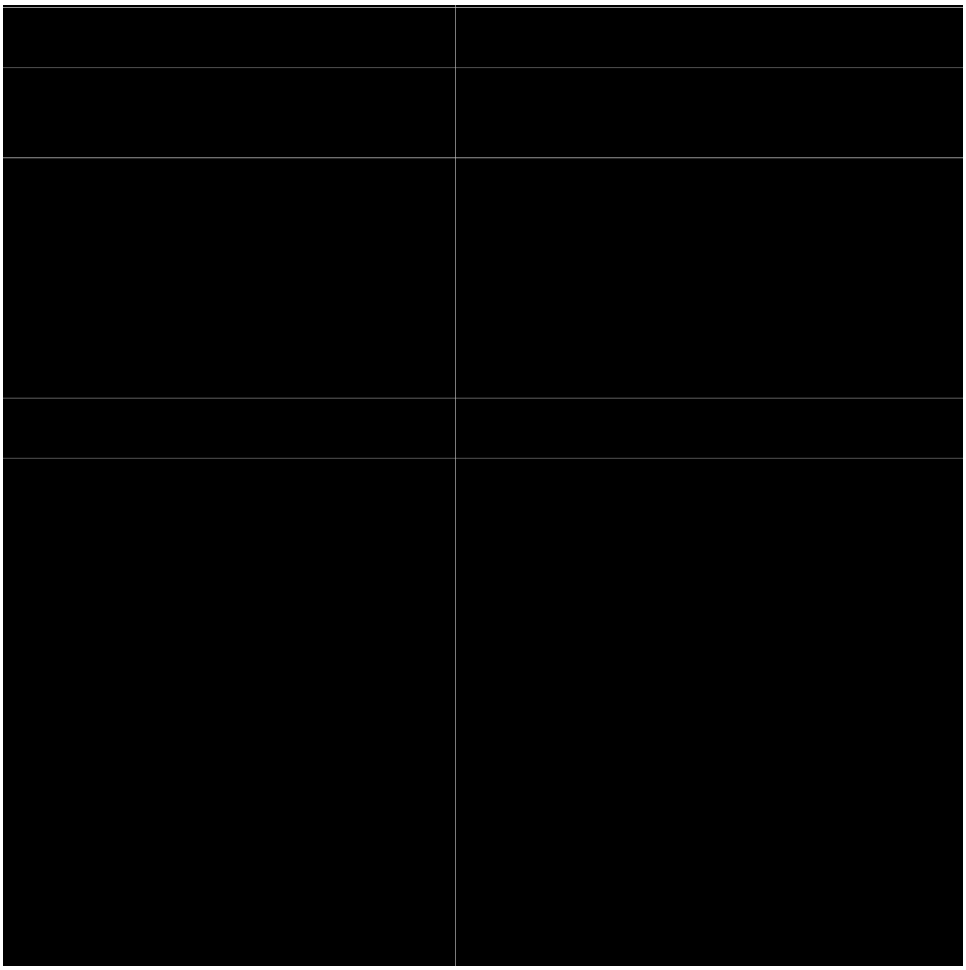


Figure 6.7: The second test image processed with mode 5.

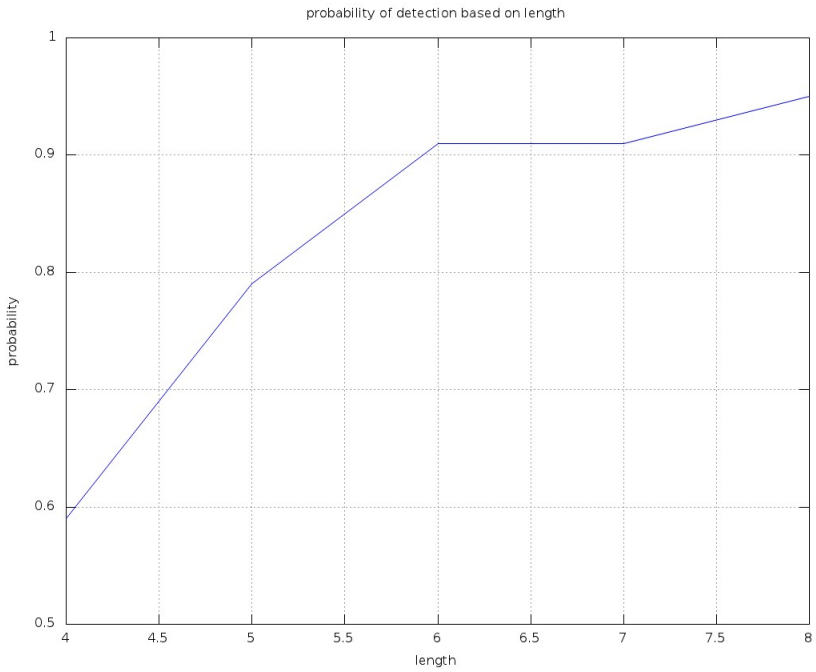


Figure 6.8: Detection probability of macrolines as a function of length. The performance is limited by random variation in orientation of microlines. See discussion in text. In this image, all background clutter was the same intensity as the target line.

This may be compared with the work of Gintautas et al., [30] who work with "amoebas" which are, by design, figures in which local microlines (our terminology) are very closely aligned. Lack of local alignment should produce a negative decision.

It is reasonably easy to see why the collinearity requirement is so sensitive. Imagine two short, nonparallel line segments one with orientation, say 53 degrees and the other 47 degrees. Suppose their centers both lie on the same 50^o line. Even though they are both have orientations close to that of the line on which they lie, when one extends the segments, we see that they are really quite different lines. The transform-based detection method we use does not distinguish distance between the line segments, so they are not considered part of the same line. It is clear from other experiments, that the human follows a curve, if the segments are touching or very close, but apparently not if the segments are not connected as in this case.

6.4 Detection as a Function of Clutter

Not surprisingly, if the background clutter is reduced, the probability of correct detection of the true macroline increases. This is illustrated in Figure 6.10.

6.5 Comparison with human experiments

In this section, the results from SLDSimulation are compared with results from [26] and [35]. Contour detection experiments with 'yes' and 'no' responses were conducted on human subjects in

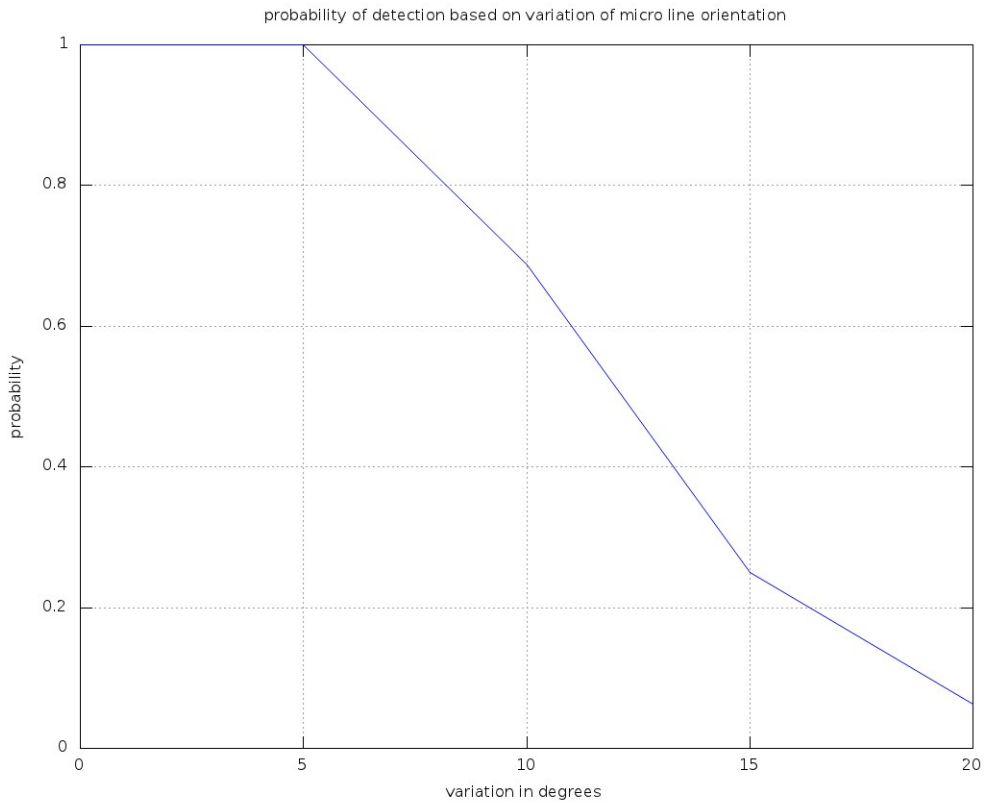


Figure 6.9: Detection probability of macrolines as a function of variance in the orientation of the microline.

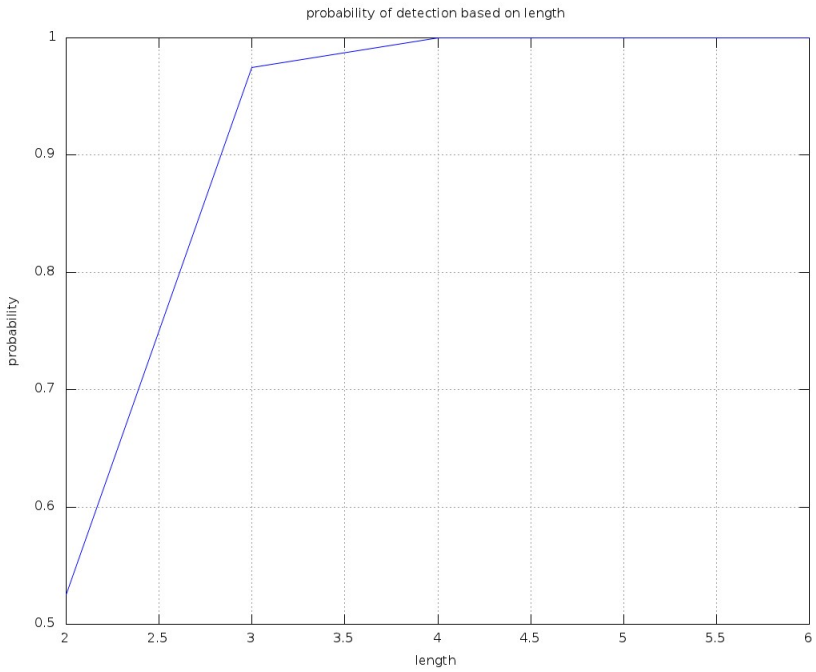


Figure 6.10: Detection probability as a function of length for a low signal-to-clutter ratio image. Here, the average brightness of a true positive pixel is double that of a clutter pixel. Compare with Figure 6.8 in which the clutter and signal are the same brightness. This is consistent with the observation at the beginning of the report that even very subtle lines may alert the line-detecting portion of the brain if the clutter is very low.

[26] and [35].

6.5.1 Similarities in experimental parameters

Here, we use the terms "microline" and "macroline" to distinguish between the short, sometimes collinear segments in the test images and the long line perceived as made from the shorter microlines. The image size used in the current set of experiments is 512 x 512, and the image is divided into squares of size 32 x 32. Every microline was constrained to fall within a square. The straight line was constructed first, and then the empty squares were filled up with microlines. These parameters are similar to those of [26], and comparable to those of [35].

6.5.2 Differences in experimental parameters

Experiments outlined in [26] and [35] involve detection of contours, while we are interested in detection of straight lines. Line segments were used in place of Gabor elements. The position of each microline was constrained to fall within a square, and the position within the square varied in Gaussian manner (with high probability of the microline falling in the center of the square), as opposed to uniform distribution commonly reported in literature. Path angle remains zero throughout, as we are interested in straight lines. It varies from element to element in the papers. Human subjects are asked "do you see a curve (or line) at all?" We ask our algorithm, "What is the orientation of the dominant line you see (if you see one)?"

6.5.3 Results Compared with Human Performance in Literature

Figure 6.11 shows the relation between variance in microline orientation along a curve as observed by humans. The microline orientation is varied randomly in a Gaussian manner. For our experiments, we consider only straight lines, and these correspond the zero angle case. (y axis of figure). It can be seen that for the zero degree case, as the variation increase from 0 degrees to 30 degrees, the percentage of correct detection by humans decreases from 100% to around 80% at 15 degrees, and to almost 50% at 30 degrees.The same trend is shown by SLDSimulation, except that we did not ask the algorithm "do you see a curve at all?"

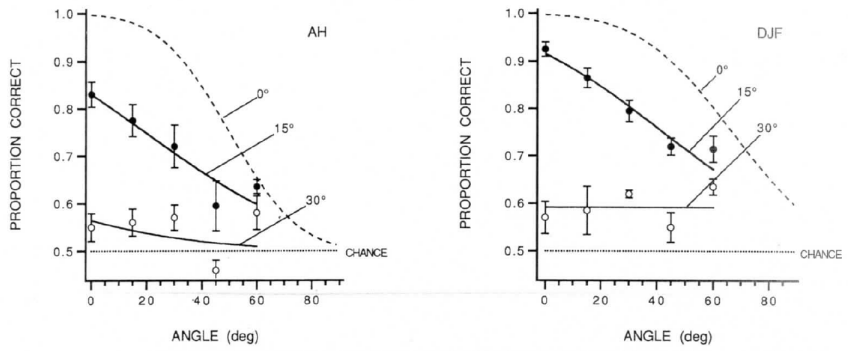


FIGURE 11. Results of Expt III. The solid symbols plot the results for orientations of the element with respect to the path (α) of ± 15 deg, and the open symbols for orientations of ± 30 deg. The dashed line shows the results from Expt I ($\alpha = 0$).

Figure 6.11: Results from experiment 4 of [26].

The set of results in Figure 6.13 are from [35]. Positional sigma refers to the sigma of the 2D Gaussian determining the position of the microlines along the path of the straight line, but constrained

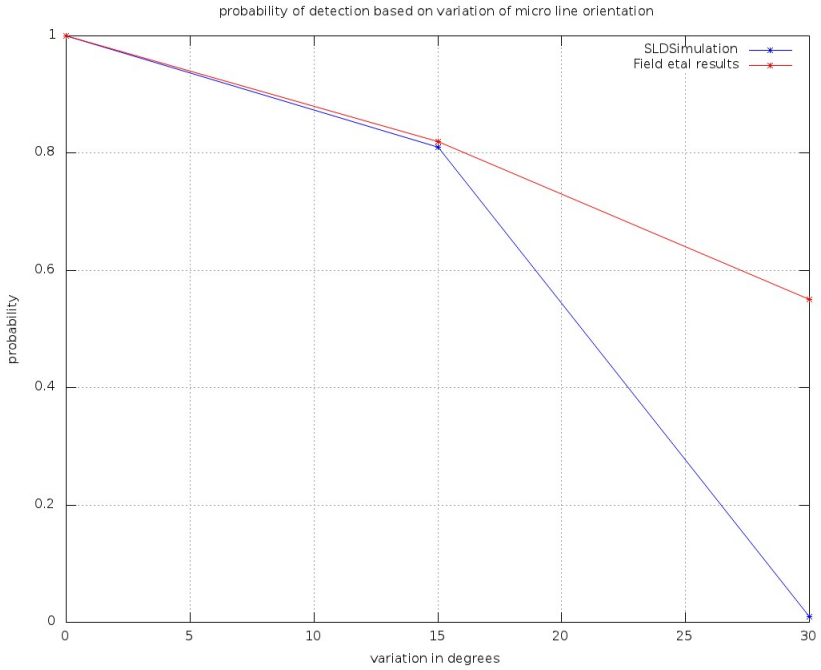


Figure 6.12: Red: results from Field et al. showing probability of detection as a function of variance in microline orientation. Blue: SLDS Simulation results.

to lie within a square. We are primarily interested in the lower left figure (unfilled circles) which reports five degree path angle contours. Hess and Dakin [35] do not report zero degree behavior, so 5 degrees is as close as available for comparison.

Figure 6.14 compares our results for straight lines with [35] at 5 degrees.

The experiments whose results are in Fig. 6.15 are similar to those in figure 6.11 but compare different parameters.

Orientation sigma refers to the sigma of the Gaussian distributed variation of microline orientation from the straight line path. The difference once again is in the path angle, which has been set to zero degrees in Fig. 6.16. The result is hence most close to foveal detection of a five degree path. Note that Figure 6.16 includes not only our results in blue but a restatement of the results from Hess [35]. If the length of the straight line had been set to 8 microlines, as is the case in [35], we would have obtained a 100% detection for the straight line by SLDSimulation. Reducing the line length to 6 affected the detection rate, and reduced it for higher sigma in the orientation variation value. Further reduction in the line length would lead to even more rapid deterioration in detection rate.

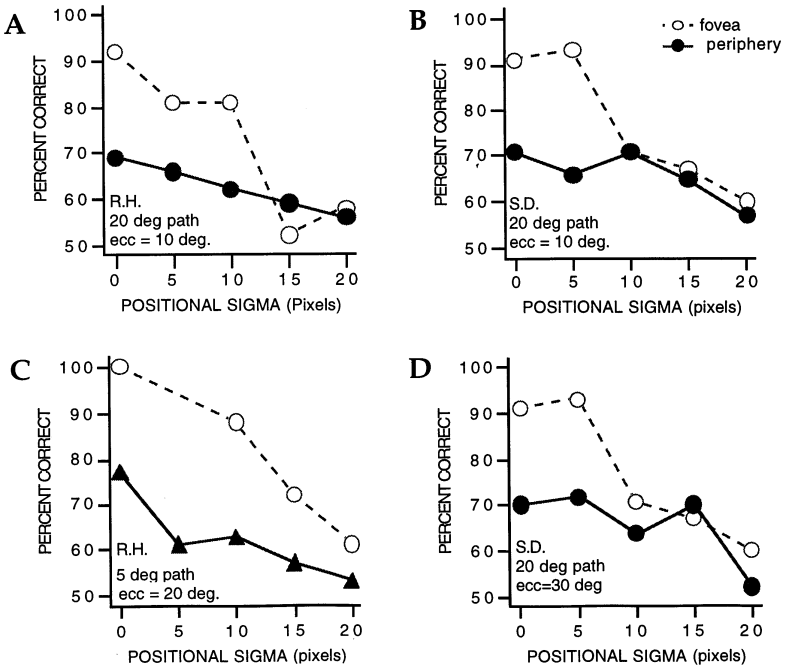


Figure 6.13: Positional Sigma results from [35].

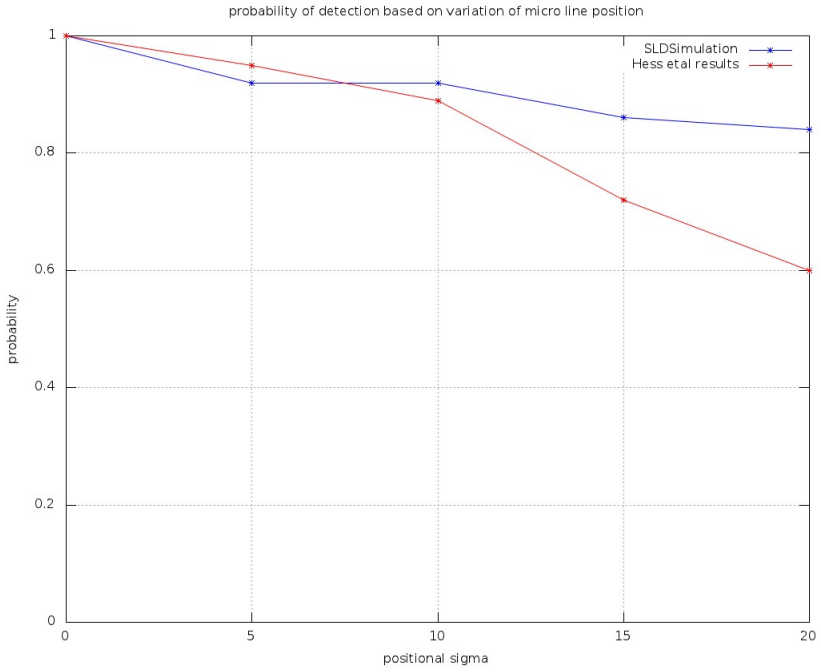


Figure 6.14: Positional Sigma comparable results.

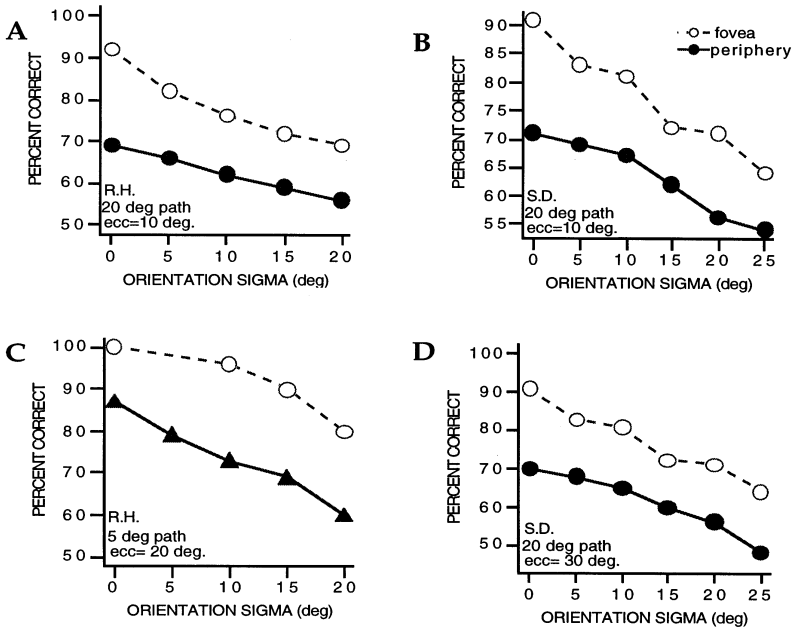


Figure 6.15: Orientation Sigma results from [35].

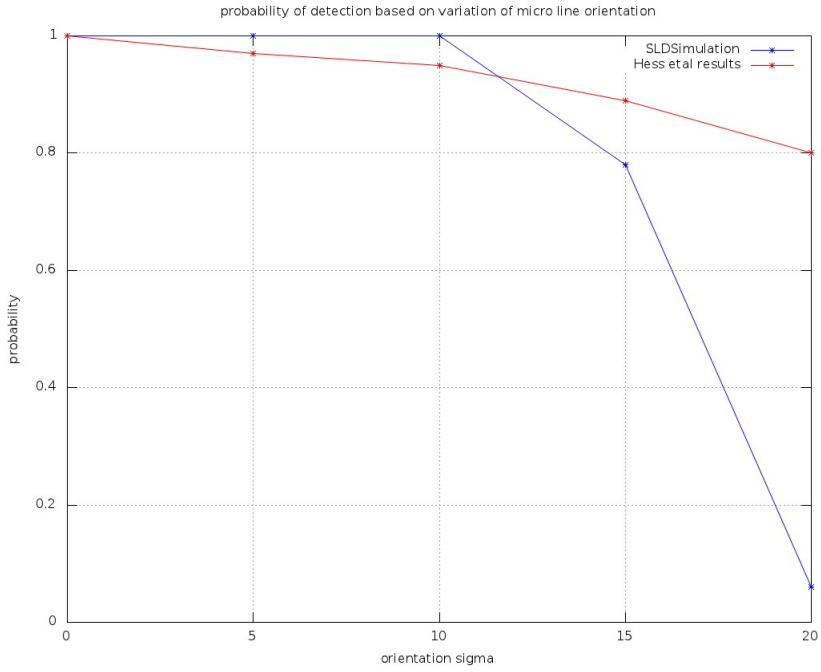


Figure 6.16: Orientation Sigma comparable results.

Chapter 7

Future Work

7.1 Correlation with Human Behavior

Our experiments in comparing human perception and simulation results can only be called "anecdotal," since we do not have the skill or facilities to design, set up, and execute experiments with humans. We need to present a human with a set of experiments similar to those described in section 6.1 and compare the results.

Further work on this project will require human experiments with an expert in human cognitive psychology. Unfortunately, there do not seem to be psychologists near NCSU who have interest or time; e.g. we located one expert at Duke University who has the skills and facilities, but he has just received significant miliary funding and will not be available for approximately two years. He suggested some other psychologists in other states who have the requisite skills. However, if we cannot identify a local collaborator, and must communicate long-distance, we might as well use people we know.

We have had a collaboration with Dr. David Badcock at the University of Western Australia who could bring to the project an expert-level understanding of both Computer Vision and neurophysiology of vision, while practicing as a cognitive psychologist with a huge experience base in experimental design. Unfortunately, travel to or from Australia would be expensive. It might be possible to perform these experiments without travel. This remains to be investigated.

7.2 Hardware

The proposed architecture requires one connection from a given *C1* cell to an accumulator cell. Each accumulator cell, however, may receive input from many *C1* cells, perhaps thousands, depending on the resolution[46]. We propose to build a small electronic model. First, the prototype will be designed using analog hardware functionally equivalent to a neural network[14, 32]. The design will be simulated on a digital computer and its performance evaluated. Then, if feasible, we will construct a hardware prototype capable of processing a reasonably (e.g. 512×512) sized image. Both analog and FPGA implementations will be evaluated.

7.3 Learning

The architecture proposed above assumes an interconnection network which is specific to detecting straight lines as they are presented in the visual field. One would suspect that such hardware is built-in, since detection of the horizon is so critical to self orientation, but biological evidence is lacking to indicate how much is built-in and how much is learned. Here, we assume the basic architecture is built in, but the weights must be learned. This philosophy is consistent with the work of Linsker [43] who assumed a locally-connected array of light detectors with local interconnections, and showed how learning based on the Hebb rule can produce receptive fields.

The Hebb rule defines how the interaction between two neurons may be associated. Given two neurons, say x and y, connected so that x provides input to y, the synapse between the neurons changes according to $\Delta_{ij} = \eta\hat{x}\hat{y}$, where $\hat{x}$ denotes the firing rate of neuron x. Thus, if firing neuron x causes neuron y to fire, then, when x fires again, y is (slightly) more likely to fire. η is the *learning rate*. Hebb learning has been described in many different variations since Hebb's original work in 1949[34]. However, when compared to more computer-appropriate algorithms such as backpropagation, it remains the most biologically-plausible. Similarly, we hypothesize that Hebb learning can result in interconnection weights which will detect long straight lines.

In this component of the research, we will ask how it is possible to arrange the accumulator array in such a way that Hebb learning can produce the interconnection weights which in turn produce the accumulation function and the peak detection function.

We propose to also use Spiking Neural Network(SNN) models for simulation of the connections between complex cells and the accumulator array. SNNs use spike trains to convey information, as opposed to neural networks which use single values[64]. The learning rule used is called spike timing dependent plasticity, and is similar to Hebbian learning. SNNs work with a large number of neurons, and may be trained with algorithms similar to backpropagation. Field programmable gate arrays may be used for their hardware implementation. To handle the large number of afferent connections from *C1* cells, a time division multiplexing architecture may be used[31].

7.4 Impact on Models for the Visual Cortex

Neurophysiological experiments have functionally described much of the mammalian visual system. The lower levels are known well, as mentioned earlier. We know, for example, that there are cells which respond consistently to specific stimuli, such as edges.

Here, we should observe that all the tests described above use line detection rather than rising edge or falling edge detectors.

At the lower levels, we also know that the cortex is roughly retinotropic. That is, cells which respond to nearby stimuli in the visual field are themselves close together in the cortex. However, other than those rather simple relationships, other researchers have not found good mapping between image features, geometry, and computational organization. The approach described here identifies very simple features, straight lines, but it potentially provides an understanding of other architectural features in the vision system.

In section 4.2 we postulated the presence of a collection of cells which function as accumulators of evidence, evidence which would suggest the presence of one or more straight edges in the scene. We have shown that the interconnections between *C1* cells and accumulator cells is not impossible, but we have not demonstrated biological evidence for such an accumulator. A significant objective of the proposed research is to address the question, "If such an accumulator exists, how can its

presence be identified and measured in a biological neural network?" Of course, we do not currently have an answer to that question, but a few directions of investigation may be suggested for future work:

- Although each *C1* cell is only required to have one afferent to the accumulator, each accumulator may have many inputs. Therefore, to keep the interconnection density down in the accumulator, some architectural variations may be provided to trade off, for example, resolution in the radial direction for interconnection density. For example, if the accumulator were represented in polar coordinates, the position of image features would be measured more accurately for features close to the center of the visual field. Could this be an explanation for the well-known "log-polar" image representation which has been detected in the mammalian visual system? Of course, the same argument for varying resolution would apply to pixels as well as to edges. In future work, it might be possible to answer this question.

- Carl Weiman published a number of papers on log-polar transform (See section 3.2), including one[62] which is particularly relevant to this proposal. In that paper, he used the duality between lines and points to show that if one simply used $\log \rho, \theta$ as the parameters of the accumulator instead of ρ, θ, the "log-Hough" array was identical to the log-polar representation of an image described in section 3.2. We propose to extend this understanding to take advantage of our earlier observation that using orientation knowledge, one can reduce the number of axons between the image and the accumulator. Weiman suggests that both may occupy the same portion of the cortex, taking care to maintain separation between image data and accumulator data. We speculate that this may lead to an understanding of the principles underlying the existence of the log-polar representation in the cortex.

- Following on the previous topic, we question how such an accumulator may be laid out in the cortex. We always draw the accumulator as if it were physically located separate from the *C1* area, but that is simply a convenience for thinking about what feeds signals to what. There is no particular reason for such a separation, and it is likely that the accumulator cells are interspersed between the imaging cells. We would hope to investigate both architectures, and evaluate average connection length, axon density, and learning/adaptation mechanisms.

Chapter 8

An Algorithm for Drawing Straight Lines

To perform the simulations described in section 6.1, it is necessary to draw straight lines. As is well-known, it is difficult to draw straight lines which are very thin (one pixel) without distortions . Figure 6.1 illustrates these phenomena, as the human detects angled lines differently from horizontal or vertical lines. It turns out that the Gabor filter which we use to estimate directional derivatives also responds differently to diagonal and vertical/horizontal lines. Correcting this is referred to in the Computer Graphics community as "anti-aliasing." The paper by Xiaolin Wu [65] probably represents the state-of-the-art in making such images look "good" to human observers.

However, in our work with Gabor filters, we found a sensitivity that humans do not seem to have. Consider a white horizontal line of length n pixels on a black background, and think of it as a rectangle 1 pixel high and n pixels long. We compare that with a diagonal (45^o) line, also n pixels long. Place a circular receptive field of diameter m pixels ($m > 2$) about both of these lines. Within the receptive field, there is more "whiteness" within the receptive field for the horizontal line than for the diagonal line, thus the Gabor filter will produce a stronger response.

Our solution to this version of anti-aliasing is as follows. To draw a line of length l, with center at point x_0, y_0 and orientation θ,

- Imagine each pixel as made up of a 9×9 array of points, equally spaced within the 1×1 area of the pixel. Refer to these points as "subpixels."
- Arrange the pixels vertically along the $x = 0$ axis.
- For each subpixel, rotate it by θ about the center of the line, and translate the resulting coordinates by x_0, y_0.
- Determine into which pixel the rotated-and-translated subpixel lies, and increment the brightness of that pixel by 1/81.

This correction improves significantly the uniformity of Gabor for lines. Figure 8.1 below, created using the new method, should be compared with Figure 6.1 which creates straight lines using simple dot-plotting.

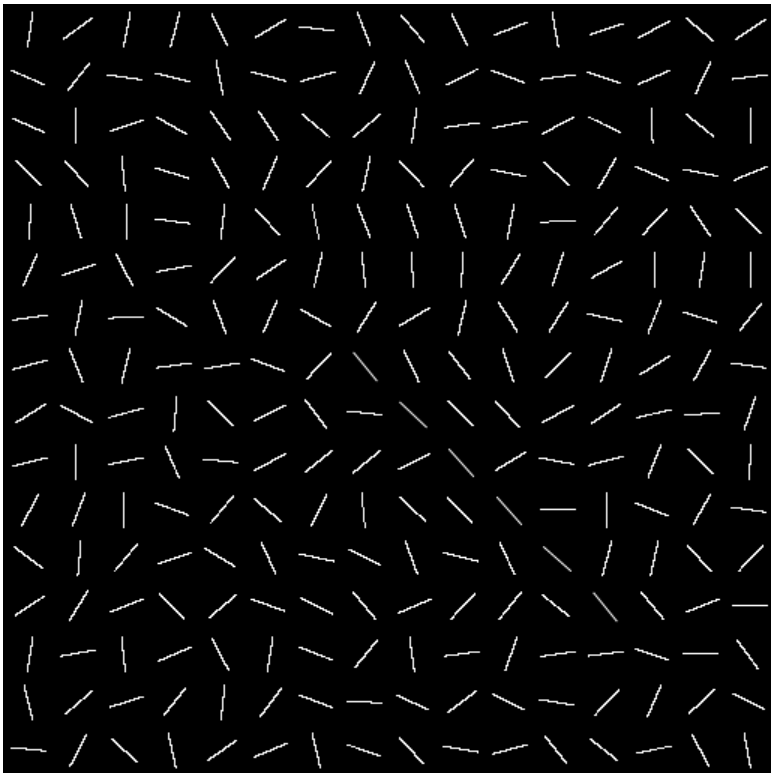


Figure 8.1: A collection of random lines segmented created using the new anti-aliasing algorithm.

Chapter 9

Conclusion

We have described an architecture for detecting linear features in images which is biologically-plausible. We have demonstrated reasonable performance estimates for this architecture through simulation. Simulation has confirmed that high resolution accumulation can be implemented with only a few directional derivatives, implemented using Gabor functions, and is therefore a feasible model for how humans detect straight lines.

If the simple, sum-of-filter output (our mode 4) is used, the accumulator peaks are roughly circularly symmetric, and have easy-to-identify peaks.

Our experiments used collinear or nearly-colinear collections of short, disjoint line segments. Disjoint line segments are much more strongly identified as belonging to a single longer line when the segments are very close to collinear, within two degrees, and the computer simulation has provided an explanation for this phenomenon in humans.

Appendix A

Determining Orientation from Four Estimates

At the $S1$ level of the standard model, we assume we have four estimates of the orientation of an edge, determined using four different Gabor filters. Here we assume that the angle of observation is or can be rotated into a frame in which two of those filters are orthogonal and we denote those as *vertical* and *horizontal*, denoting them as basis vectors $[0\ 1]^T$ and $[1\ 0]^T$ respectively. The remaining two filters are also mutually orthogonal, and are represented (in the frame of the other two – the base frame) by $[\ \frac{1}{\sqrt{2}}\quad \frac{1}{\sqrt{2}}\ ]^T$ and $[\ \frac{1}{\sqrt{2}}\quad -\frac{1}{\sqrt{2}}\ ]^T$ respectively. The direction of the edge in the base frame is the unknown ordered pair $G = [G_x\ G_y]^T$, and this direction has been measured four times, producing $\mu = [\ \mu_1\quad \mu_2\quad \mu_3\quad \mu_4\ ]^T$.
Constructing the matrix B from the four basis vectors produces the relation

$$BG = \mu\ , \tag{A.1}$$

Using the pseudoinverse to find the minimum squared estimate of G for this overdetermined problem,

$$G = (B^T B)^{-1} B^T \mu \tag{A.2}$$

Once we have the vector G, those two numbers define the direction.

$$(B^T B)^{-1} B^T = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \\ \frac{1}{2} & 0 & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} \end{bmatrix} \tag{A.3}$$

Thus, from four measurements of edge direction, a least-squares estimate can be derived by a simple sum-of-products.
The two values of G can then be fed into a quantization neural network to obtain a single signal which indicates the direction (angle). The quantization neural network is basically a straightforward lookup table, with n inputs and 2^n outputs, selecting one output axon as a function of the code on the inputs; functionally a binary decoder.

Appendix B

The Log-Polar Transform

The material in this appendix was written by Theju Jacob as a supplementary projected related to this project.

B.1 Introduction

In this report we examine the nature of computation in the visual cortex as discussed in literature. A log polar transform is believed to be the mathematical mapping which can explain how signals received by the eye are interpreted by the human brain. We start with a brief introduction to various elements in the human visual pathway. Beginning with experiments which provided the first clues to such a mapping, we trace literature which first put forth the log polar idea, and then at various modifications made to the original idea when more experimental results became available. We also look at the two major explanations given for such a structure in the brain, and at the arguments for and against each. Section 2 looks at literature primarily from the 1960s and 1970s. Papers from 1980s, 1990s and 2000s are examined in section 3. We ponder the question of which of the two explanations might be more accurate, and make our concluding statements in section 4.

The human visual system enables us to process and respond to all kinds of visual stimuli. Various structures along the visual pathway, shown in Fig.1, constitute the visual system. Starting at the retina, neurotransmitters travel through the lateral geniculate nucleus to reach the visual cortex. The retina itself is a complex structure, with rods and cones acting as photodetectors, and layers of horizontal, bipolar, amacrine and ganglion cells above them before optic nerve fibres are reached, as in Fig.2. We have the fovea located almost at the center of the retina, representing about 5° of visual angle. The highest concentration of cone cells and a near absence of rod cells are found at the fovea. The outer region surrounding the fovea is called perifovea.

The visual cortex is divided into several areas named V1, V2, V3 and V4, Fig.3. In this study, we are primarly concerned with V1, also called the primary visual cortex or striate cortex, though other areas do make an appearance at various instances.

B.2 Beginnings - 1960s,1970s

The widely accepted notion today is that the projection from the mammalian retina to the visual cortex takes the form of a logarithmic conformal mapping. A conformal mapping is a transformation that preserves local angles. The idea of a logarithmic mapping appeared in literature for the first

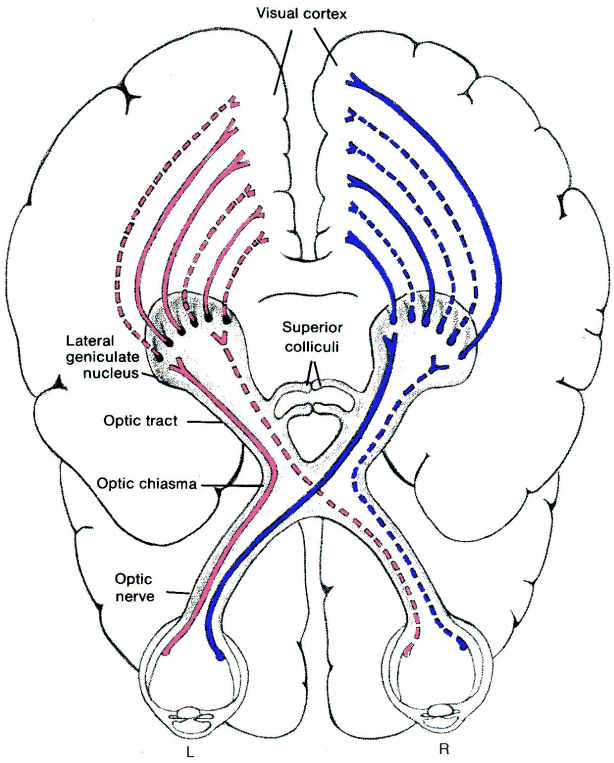


Figure B.1: Visual Pathway Structure from [28]

time in 1977[52]. The results of physiological experiments from previous decades are outlined and the reasons why a logarithmic mapping explains the results are put forward by Schwartz in [52].

Results from [18], [39], [40], [1] and [2] are mentioned extensively in [52]. In [18], Daniel and Whitteridge outline experiments involving insertion of electrodes into brains of monkeys and baboons, and discuss the responses to various visual stimuli. The authors plotted the paths traced in the visual field by the electrodes in various cortex locations. The magnification factor, which is the ratio of linear distance in mm between two points on the cortex to the angular distance in degrees of corresponding points in the visual field, was computed and plotted for different segments of the visual field. It was found that the magnification factor was the same for all points with the same radius in the visual field, regardless of the angular coordinates. A plot of the results obtained by the authors is shown in Fig.4, where eccentricity is the angular distance in degrees from the center of the visual field. We can see that the magnification factor decreases as eccentricity increases. However, the relationship is clearly not linear.

In [39] and [40], further studies of the monkey striate cortex are outlined by Hubel and Wiesel. They detail orientation columns and ocular dominance columns and their mutual relationship. Orientation columns refer to groups of cells located close together, which show a preference for a particular orientation, while ocular dominance columns refer to groups of cells which show a preference for a particular eye. The authors inserted probes into the cortex and noted how the orientation and ocular dominance varied across the cortex. They note that the total width between an array of orientation slabs that take in all of 180 degrees of orientation and a left plus right set of ocular dominance columns amount to 0.5 to 1 mm in the cortex. Various plots outlining the changes are outlined in their work. A plot relevant to our current investigation, showing the variation in

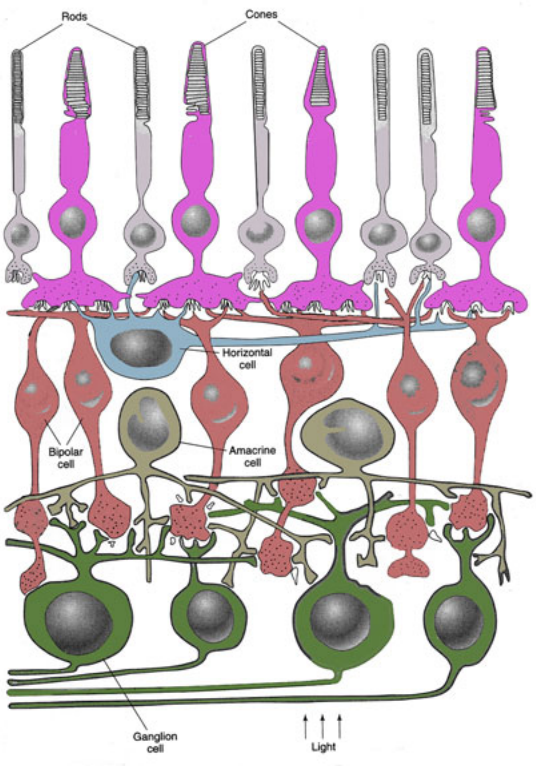


Figure B.2: Cells in Retina from [28]

the receptive field size and magnification factor with distance from the fovea is shown in Fig.5. We note that the receptive field size as well as the inverse of the magnification factor increase with an increase in the distance from the fovea. A more detailed explanation of these results and other results related to the mammalian visual system can be found online in [37].

In [1] and [2], Allman and Kaas look at the representation of the visual field in V2 as well as in the medial area of the visual cortex in owl monkeys. Recordings were done inserting microelectrodes into the owl monkey cortex and stimulating the eye using bars of light. They discovered that the central 7° of the horizontal meridian correspond to a zone of the cortex across the width of V2, after which the horizontal meridian representation splits and forms the anterior border of V2. The vertical meridian was found to be covered by the posterior border of V2. They also found the areas in V2 which represented the central sections of the visual quadrant. The medial area was found to devote a greater proportion of its area to the peripheral areas of the visual field.

The results from Daniel and Whitteridge[18] were further analyzed by Schwartz [52], and the following mathematical expression was given for cortical magnification:

$$\mathrm{m} = \mathrm{k}/\mathrm{r}$$

where m is the magnification, k is a constant, and r represents eccentricity from foveal representation.[52] states further that magnification is a differential quantity, and that we are interested in an analytic function whose derivative is radially symmetric and is proportional to 1/r. The following complex logarithmic function is put forth as satisfying the before stated requirements.

$$\mathrm{w} = \log(r\mathrm{e}^{i\phi})$$

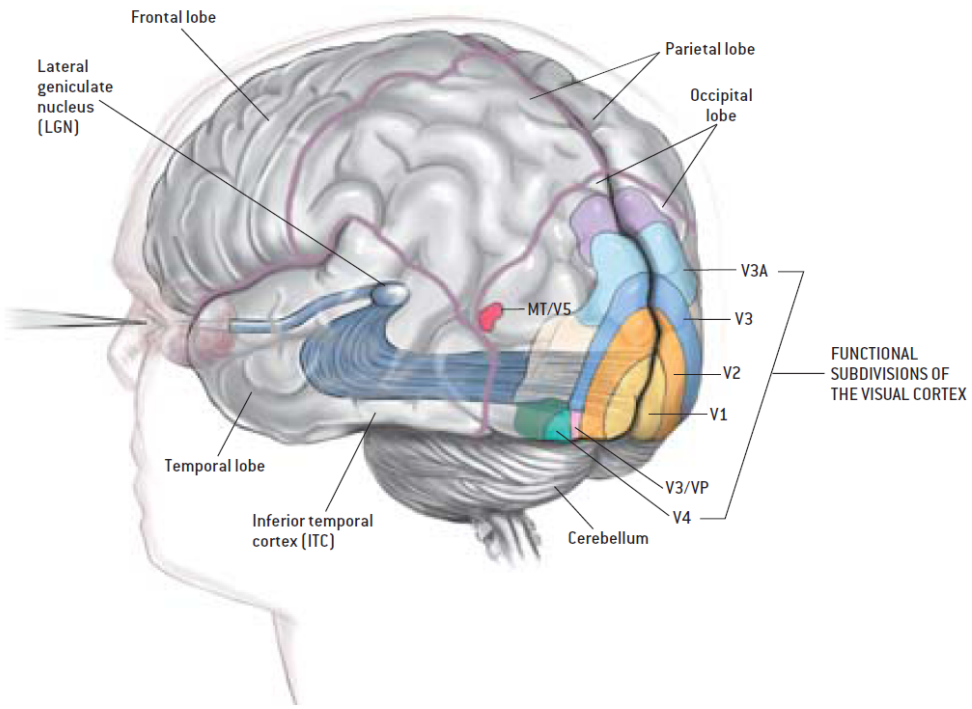


Figure B.3: Visual areas from [44]

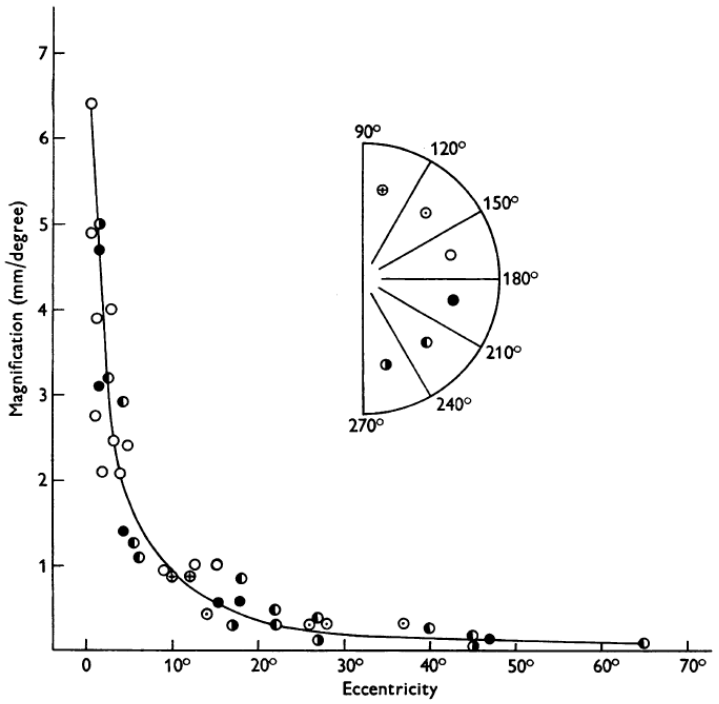
where (r,ϕ) represents a point in the visual plane, and complex number w represents a point in the cortical plane.

The magnification factor for the lateral geniculate nucleus or LGN takes the same form as that for the striate cortex. The density of retinal ganglion cells as well as reorganization of neurons while projecting from LGN to striate cortex are both thought to bring about the final form of the striate cortex retinotopic map. The structure in the secondary visual cortex is also shown to be described by a logarithmic conformal mapping. The authors also discuss how such a mapping can lead to the structural regularity observed in the striate cortex by previous works [39] and [40]. In particular, equal angular steps in the visual plane must transform to equal linear steps in the cortex, and the complex logarithm is shown to accomplish such a transformation.

In [63], log spiral grids are put forth for digitization of images. Rotation and magnification of images become simple pixel shifts on using the described technique, thus saving computation steps and memory. The authors describe various methods to achieve the necessary transformation of the image grids by introducing various tessellations.

B.3 1980s,1990s,2000s

In this section, we continue with the investigation of the presence of log polar mapping in the human visual cortex. In [53] Schwartz states that the local columnar structure as well as the retinotopic mapping found in the cortex can be explained by means of complex logarithmic mapping. The author also suggests a logarithmic mapping as a mechanism for size and rotation invariance in



Text-fig. 4. The magnification factor plotted against eccentricity. The data have been grouped in sectors each of 30°.

Figure B.4: Plot of a set of results from [18]

vision. The expression derived previously in literature has now been modified to compensate for the divergence at zero to:

$$\mathrm{w} = \log(r\mathrm{e}^{i\phi} + \mathrm{a})$$

where a is a constant. The modified expression gives us a linear map for small values of $r\mathrm{e}^{i\phi}$ ($<$ a) and a logarithmic map for larger values. The author also demonstrates how changes in size or rotation of objects reduce to a linear shift under the mapping.

Schwartz et al. in [54] studied the role of inferior temporal cortex rather than the primary visual cortex. The IT cortex is thought to play a crucial role in visual object recognition, by enabling us to identify shapes and patterns. IT neurons have large receptive fields that extend to both half visual fields, and includes the center of gaze. Five macaque monkeys were tested with slits of light, complex objects and Fourier Descriptor(FD) [3],[66] stimuli. The FD for a particular shape was obtained by expanding the boundary orientation function as a Fourier series. The boundary orientation function of a shape in turn is determined by measuring the orientation of the shape's boundary at regular intervals around the perimeter. Using FD to describe a shape would make it invariant to position and shape of the stimulus. For the majority of IT neurons thus tested, size, position and contrast variations did not affect the output, which appeared to suggest that the cells must be sensitive to the overall shape of the stimulus. The authors also suggest that coding of boundary curvature may not be the exclusive function of IT neurons, as some of them have been found to be selective for color, texture and spatial frequency.

The striate cortex responses to visual stimuli in rhesus monkeys are recorded in [22] by Dow et al., and the relationships between eccentricity and magnification factor, receptive field size and

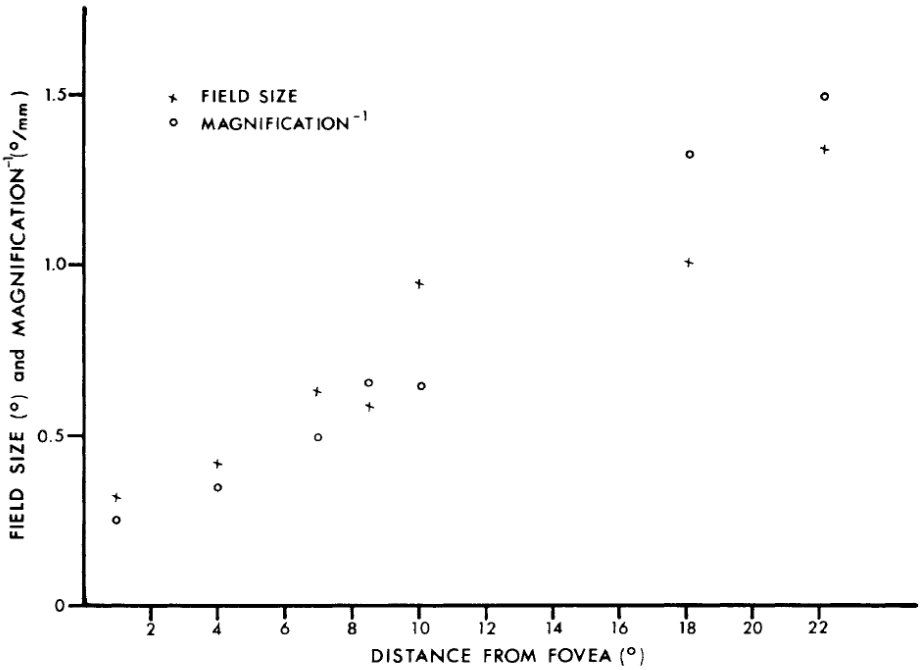


Fig. 6A Graph of average field size (crosses) and magnification^{-1} (open circles) against eccentricity, for five cortical locations. Points for 4°, 8°, 18° and 22° were from one monkey; for 1°, from a second. Field size was determined by averaging the fields at each eccentricity, estimating size from (length × width)$^{0.5}$.

Figure B.5: Plot of a set of results from [40]

the minimum angle of resolution are examined. The authors found that their results agreed with previous literature [18],[39],[40] with minor modifications, except that they did not find inverse magnification to be related to the minimum angle of resolution by a constant of proportionality, like in [18]. They also note that the parallel relationship between field size and inverse magnification do not hold well in the fovea, as inverse magnification becomes smaller than field size at an eccentricity less than 5°. The same authors present a detailed mapping of the monkey striate cortex in [23]. They describe a new technique for transferring various penetration sites on the cortex onto a map, so as to form a coherent picture of the cortex and its various areas. Their technique enabled detailed comparisons of different hemispheres of the cortex. Parameters measured include vertical/horizontal magnification anisotropy, which was found to be 1.5:1 at central eccentricities, and foveal magnification, which was found to be different along different meridians in different fields.

B.3.1 Emphasis of Central Vision in Retina

The hypothesis that higher retinal ganglion cell densities lead to the final form of striate cortex retinotopic map was previously seen in literature[52]. A strong argument for higher density of ganglion cells in the fovea leading to higher emphasis on central vision in the cortex is put forth in [61] by Wassle et al. Those authors used improved techniques for determining ganglion cell densities in monkeys. Amacrine cells, known to be displaced regularly into the ganglion cell layer, were discounted. Results are shown in Fig.6. The density of cones were found to decrease from a peak of 250,000/mm^2 at the fovea to 11,500/mm^2 at 3 mm eccentricity. Cone pedicles, the synaptic terminals of cones, are nearly absent at the fovea, but increases sharply and flattens at

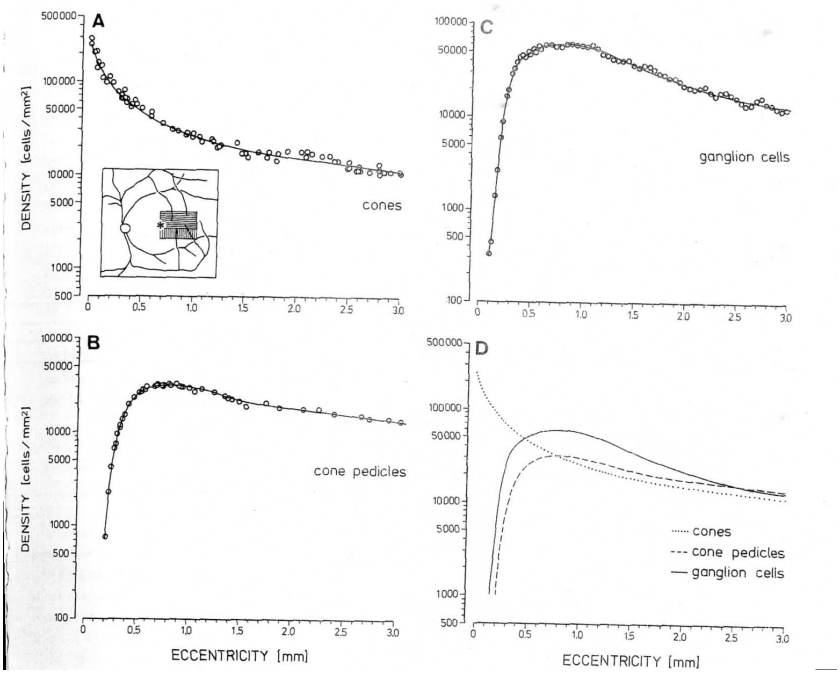


Figure B.6: Plot of a set of results from [61]. From [61] - 'Density gradients of cones, cone pedicles and ganglion cells along the temporal horizontal meridian of a macaque monkey fovea. The inset in (A) shows the retinal area examined. The blind spot(open circle), the fovea(asterisk), and some blood vessels characterize the central retina. A rectangl 3.5 x 2.5 mm wide was cut from the eye cup, vertical sections were cut through the upper part, and horizontal sections through the lower part. (A)Cone density; (B) cone pedicle density; (C) ganglion cell density; (D) comparison of the density gradients (modified from Wassle et al., 1989).'

500μm, peaking at a plateau of 32,000/mm^2. Ganglion cells have a density profile similar to that of cone pedicles, but more ganglion cells are present at lower eccentricities. A plateau is present at 800μm, with a maximal density of 60,000/mm^2. The authors point out that the change in ganglion cell density follows a profile similar to change in magnification between central and peripheral visual field in the cortex, and hence they claim that ganglion cell density can fully explain the variation in the cortical magnification factor.

B.3.2 Emphasis of Central Vision in Retino Cortical Pathway

Van Essen et al.[60] undertook a detailed study of the visual cortex in macaque monkeys. A smooth variation of recording sites within the cortex was found to correspond to a smooth progression in receptive fields, thus preserving all neighborhood relationships. This further implied that the cortex is retinotopic. Large variations were found in the relative emphasis on different parts of the visual field representation in striate cortex between individuals. The authors also state that the striate cortex lacks radial symmetry in 2 respects - greater emphasis is placed on the horizontal meridian when compared to the vertical meridian, and the inferior parts of the visual field are slightly more emphasized than the superior parts of the visual field. Plots for magnification factors vs eccentricity were obtained, as shown in Fig.7. The authors also gave new expressions for linear and areal cortical magnification factors which better fit the data than those previously available in literature. For

example, the areal cortical magnification factor, the ratio of area in the cortex in mm^2 to the corresponding area in the visual field in deg^2, was now given by

$$M_a = 140(0.78 + \mathrm{E})^{-2.20} \mathrm{mm}^2/\mathrm{deg}^2$$

where E is the eccentricity. The equation for M_a predicted a ratio of 4400 in areal magnification at 1° vs 80° eccentricity. On the other hand, retinal ganglion cell density at 1° was found to be about 50 times that of the density at 80°, which implied that central vision is emphasized much more in the cortex than in the retina.

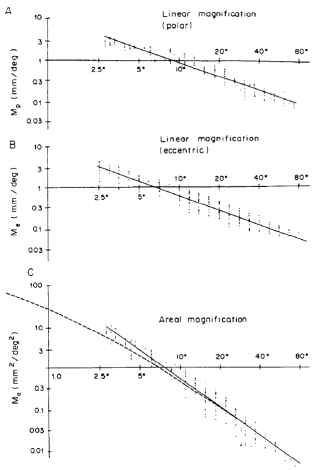


Figure B.7: Plot of a set of results from [60]. From [60] - 'Cortical magnification as a function of eccentricity. Linear magnification (mm/deg) along iso-polar contoursis shown in (A) and along iso-eccentricity contours in (B). Areal magnification is shown in (C). Calculation of magnification factors was based on the topographic contours illustrated in Fig.5. The map coordinates of the intersections of iso-eccentricity and iso-polar contours were entered into the computer via a graphics tablet. In regions where there was sufficiently detailed information about visual topography, the spacing of contours was twice as close as that shown in the preceeding figure. Regression lines were calculated according to the least-squares method. For each compartment the length of each side was divided by the length of the corresponding visual field compartment to obtain the linear magnification factors (M_e and M_p , and the area of the cortical compartment was divided by the area of the visual field compartment to obtain the areal magnification factor.'

Azzopardi and Cowey[4] further supported the idea that the central visual field is emphasized more in the cortex than in the retina. The authors used a retrograde transneural tracer on rhesus monkeys, and the number of ganglion cells projecting to marked areas of the cortex were counted. They computed the average area in the visual cortex receiving a projection from a single ganglion cell, and used that as an index of cortical allocation. If cortical allocation were to mirror the ganglion cell distribution across the retina, the ratio of perifoveal to peripheral cortical allocation should be 1. However, the ratios were found to lie in the range from 3.29 to 5.93. Therefore, the authors state, the emphasis on central vision in the cortex cannot be explained by the changes in retinal ganglion cell density alone. They hence contradict the results from [61].

The same authors conducted more detailed experiments on macaque monkeys and presented their results in [5]. This time, they studied the allocation of neurons for the fovea in the dorsal lateral geniculate nucleus as well. Once again, cortical allocation values larger than 1 were obtained,

thereby implying that the cortical map of the retina is not peripherally scaled, but that relatively more cortex per ganglion cell is alloted in the fovea and surrounding retina.[5] also states why arguments about estimates of ganglion cells representing the fovea being erroneous do not hold in the current set of experiments - the authors are comparing cortical areas with their total number of ganglion cells, including laterally displaced ganglion cells. Arguments for higher ganglion cell counts near the fovea, as well as for the presence of displaced amacrine cells in the ganglion cell layer thereby leading to erroneous ganglion cell count have also been discounted. The results obtained by the authors indicate that the representation of the fovea is magnified when moving from the retina to the lateral geniculate nucleus and from the lateral geniculate nucleus to the striate cortex.

Further support to the idea of an overrepresentation of the fovea in the cortex is given by Popovic et al[48]. The authors analyzed data from already available literature and computed the relationships between cortical magnification factor, effective ganglion cell separation and the minimum angle of resolution. A set of results are shown in Fig.8. A linear relationship was found between effective ganglion cell separation and minimum angle of resolution. They show that the inverse magnification factor and the before-mentioned quantities of ganglion cell separation and minimum angle of resolution are also related linearly. The results point out that a markedly larger amount of striate cortex per cell and amount of visual cortex is devoted to fovea and surrounding retina. The paper concludes that the overrepresentation of the fovea and immediately surrounding retina are caused by an additional magnification in retino cortical pathway.

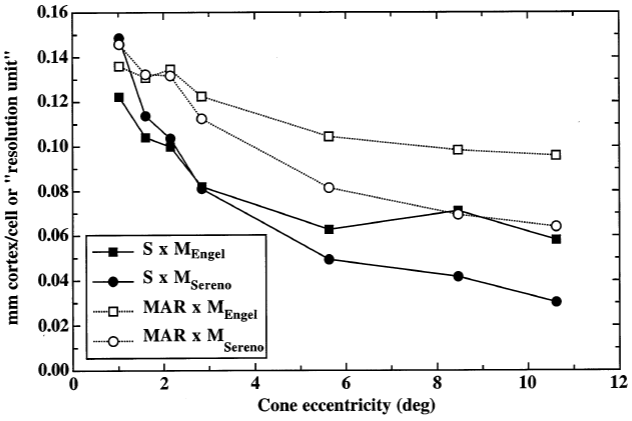


Figure B.8: Plot of a set of results from [48]. From [48] - 'Plot of the product of effective GC separation (S,deg) and the linear cortical magnification factor (M,mm/deg) vs. eccentricity (filled symbols), as well as the product of resolution thresholds (MAR, deg) and M vs. eccentricity (open symbols), for the data of Engel et al.(1997) Sereno et al.(1995), describing the reserved amount of visual cortex (in mm) per GC and the amount of visual cortex needed to process a given resolution threshold, respectively.'

B.3.3 Later studies, use of fMRI

The question of how information transfer happens between neurons and how it is influenced by cortical organization is examined in [58]. There are 2 possibilities for how information transfer occurs between neurons: information is present in the spike rate of neurons, or in the precise timing of individual spikes. The authors examined excitatory post synaptic potentials (EPSPs) and

inhibitory post synaptic potentials (IPSPs) of neurons and the question of how they are balanced. For orientation selective neurons, for example, EPSPs and IPSPs occur frequently for optimally oriented stimuli. As excitatory inputs bombard a cell in the hundreds, much more than necessary to depolarize a cell membrane from resting potential to spike threshold, the authors propose that the primary role of inhibition is to prevent the cell firing rate from reaching saturation. As excitatory synapses outnumber inhibitory synapses by 6:1, the authors suggest that inhibitory synapses might be more effective in their activity than excitatory synapses. In summary, the authors establish that 'a random walk model with balanced excitatory and inhibitory inputs allows the neuron to behave as an integrate and fire device and maintain a reasonable response rate, with irregular ISI'.

The use of functional magnetic resonance imaging to study the human visual cortex directly is seen in subsequent papers. In [24], Engel et al. used a contrast reversing checkerboard as stimuli. As the stimulus moved from fovea to periphery, the fMRI signal was delayed at locations containing neurons with peripheral receptive fields when compared to those containing neurons with foveal receptive fields. A rotating wedge stimulus was used to study the retinotopic organization with respect to polar angle. They also obtained clear demarcation between various areas in the visual cortex. Similar stimuli were used in all subsequent fMRI studies by other authors as well. A study similar to [24] of the human visual cortex using fMRI was conducted by Sereno et al. in [56]. The extent and borders of various visual areas were identified, and iso-eccentric and iso-polar maps of the visual areas obtained. A comparison between human and other primate visual areas is made, and the cortical magnification factors plotted, as shown in Fig.9. Their studies imply that humans have extreme emphasis on the center of gaze, however, they leave the question of the reason for the emphasis (emphasis on the retina vs emphasis on the cortex) open.

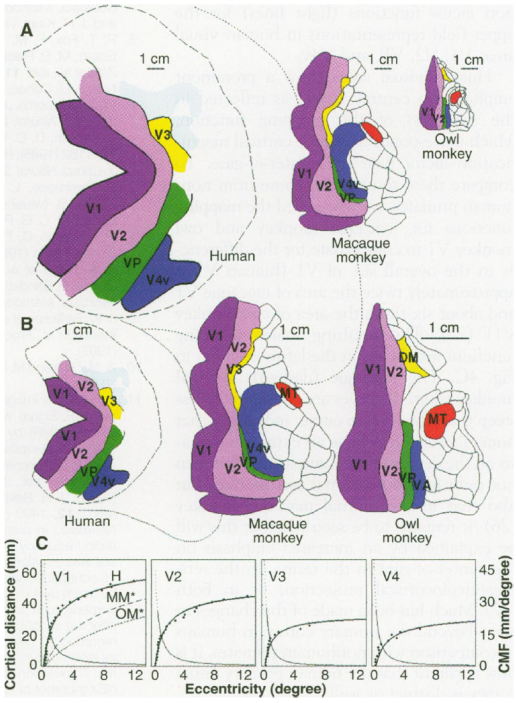


Fig. 4. Schematic summary of retinotopic visual areas in the owl monkey (*7*), the macaque monkey (*21*), and the human (present study) at the same scale (**A**) and approximately normalized by the area of V1 (**B**). (Human V1 is twice the area of macaque V1, with larger ocular dominance columns and cytochrome oxidase blobs, but a similar number of cells.) Visual areas in humans show a close resemblance to visual areas originally defined in monkeys. The anterior border of the visual cortex in humans was estimated by using the superior temporal sulcus and intraparietal sulcus as landmarks. In (**C**), the mapping functions (heavy lines; scale is on the left axis) and magnification factor functions (light lines; scale is on the right axis) are shown for the upper field representations of human V1, V2, VP, and V4 (*24*). The V1 mapping functions for owl monkeys (OM*, dotted) and macaque monkeys (MM*, dashed) shown at the left were scaled up to match the overall size of human V1 (*25*). An increased emphasis on the center-of-gaze in human V1 is evident. CMF, cortical magnification factor.

Figure B.9: A set of results from [56]

Balasubramanian et al.[6] modifies the previously proposed expressions for log polar mapping with the idea of a dipole mapping:

$$w = \log(z+a) - \log(z+b)$$

where w is a point on the cortical plane, z is a point on the visual field and a and b are constants. The authors point out that the modified expression captures the shape of the V1 boundary exhibited at peripheral representation as well as the fact that inverse cortical magnification factor is sub-linear in the peripheral field. The dipole mapping is then further modified in order to explain more complex features like the boundary conditions between V1, V2 and V3 and the spacing of iso-eccentricity lines.

Dougherty et al.[21] describes the use of fMRI to study visual cortex areas V1, V2 and V3. The authors obtained maps of the locations of these areas in the cortex, and determined their surface areas for the central 2°-12° of the visual field. For the central 1°-2°, the various areas were not distinguished, and the combined surface area for this central region of the visual field was determined to be 2100 mm^2. The variation of cortical magnification with eccentricity was found to be in agreement with previous literature. The plots showing the variation for all three visual areas, for all of the test subjects, were obtained, Fig.10.

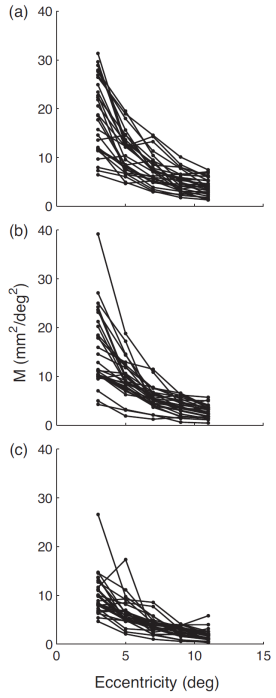


Figure 4. Surface area cortical magnification functions for all hemispheres. (a) V1, (b) V2 and (c) V3.

Figure B.10: A set of results from [21]

Further studies of the human visual cortex using fMRI are seen in [50] by Schira et al. The authors determined more accurately the borders between V1, V2 and V3, and obtained 2 dimensional reconstruction of various iso-polar and iso-eccentricity lines in V1. They started out with the following expression for magnification:

$$m = k/(r+a)$$

where m is the magnification at eccentricity r, and k and a are constants. They determined the value of a to be approximately 0.75 for both V1 and V2, and k to be 1.92 for V1 and 1.59 for V2. The magnification vs eccentricity plots obtained are shown in Fig.11. The authors also modified the mapping introduced in [6] so as to preserve the meridional isotropy of the area at the expense of small degree of local anisotropy. The question of whether V1 is actually flat, and if variability between individuals can affect the construction of a generic model for V1 is also examined in the paper. The authors make the case for V1 being intrinsically flat, and state that a general model is the best way to estimate V1 parameters, in the cases where we are unable to measure V1 morphometry for individuals.

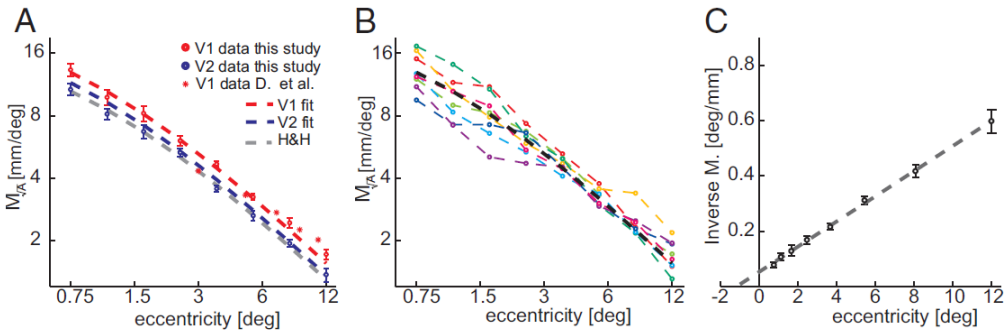


Figure B.11: A set of results from [50]. From [50] - 'Mean $M_{\sqrt{A}}$ vs.eccentricity across all subjects fitted by Eq.1. A: measurements across areas and studies. V1 data are plotted in red(fit: a = 0.77 ± 0.03;k = 19.2 ± 0.25 and V2 data are plotted in blue(fit: a = 0.73 ±0.03;k = 15.9 ± 0.21). The gray line shows the cortical magnification function(V1) from Horton and Hoyt (1991a)(a = 0.75;k=17.6) and the red * symbols are points from Dougherty et al.(2003), error bars depict SE. Because our measurements did not extend beyond 12° of eccentricity, the data are well described by the monopole model or the corresponding linear magnification function(Eq.1).B: individual measurements from the present study. Colored graphs show the results in individual subjects, the dotted gray line the fit for V1. C: mean inverse magnification of V1 with the best-fitting straight line for ready comparison with studies such as Rovamo and Virsu (1979, Levi et al.(1984), McKee and Nakayama (1984), Stensaas et al.(2001), Tyler (2001), and Duncan and Boynton (2003).'

Boucart et al.[15] conducted a study with the following purposes - 1)compare the performances in implicit and explicit recognition tasks using same set of stimuli at 2 peripheral eccentricities - 30° and 50° 2)Assess whether people who have lost central vision early in life develop their peripheral vision. Implicit recognition involved facilitation for old stimuli as compared to new stimuli in terms of latency and accuracy. Images were presented in 2 stages - a test stage, and a study stage. In the study stage, the images presented included - 1)images previously presented in the study stage, 2)images of objects with same names but different appearances as the images in test stage - for example, the image of a different cat 3)images of new objects. The authors found no significant difference in performance between identical and same name pictures in people with normal vision. New pictures did poorly when compared to identical and same name pictures in terms of response time and accuracy. For candidates who had lost central vision, identical pictures were identified faster, but no significant difference was found between performances for new pictures and same name pictures. In people with central vision loss, the visual system appeared to have put its resources into developing a preferred retinal locus which would stimulate cortical regions which previously responded to centrally displayed stimuli.

We also came across papers on contour detection during the course of our investigation. In [27], Field et al. present their results from 5 experiments which tested the ability of observers to identify contours from within a field of randomly oriented elements. The experiments were conducted by varying orientation, spacing and so on between the elements belonging to the contour. Their results led the authors to propose a local association field in our visual field. In [30], Gintautas et al. measured time taken by human observers to identify segmented, closed contours in backgrounds with clutter, and found that it matched their model of local association fields in the cortex. When considering probabilistic methods for edge detection, it has been found that there is a local maximum in the pairwise probability distribution for edge elements that are nearly co-circular as well as nearly parallel. The experiments in [29] suggest that humans have good knowledge of the pairwise statistics of edge element geometry and contrast polarity, and that they use the knowledge efficiently. Feldman[25] puts forth a Bayesian model for contour detection, and suggests that humans detect contours using such a model. In [45], Ma et al. suggest that humans take into account the reliability of the observation when trying to detect targets, and they put forward a log likelihood ratio to capture the idea.

B.4 Conclusion

In the literature we came across, far more papers support the idea that the fovea is indeed overrepresented in the cortex, and that this cannot be explained by the higher number of cones/ganglion cells found in the fovea and surrounding areas alone [60],[4],[5],[48],[24]. We support the same hypothesis. Fovea, which occupies only 0.01% - 0.025% of the area of the retina, is represented by approximately 8% of the striate cortex area, and this emphasis on central vision does not directly correspond with the emphasis in the retinal cell density. Hence, the additional magnification must happen in the retino cortical pathway, more specifically in the lateral geniculate nucleus [5]. We put forth the idea that the additional emphasis for central vision in the cortex is the result of a learning process by the visual system, and that this learning was triggered by the higher number of cones/ganglion cells in the fovea we started out with. Indirect support to this idea can be seen in [15]. In the absence of central vision, a preferred retinal locus which triggered the original central vision cortical regions developed in the test subjects. So in a person who starts out with normal central vision, the fovea and surrounding areas would get such a preferential treatment. The reason for such a preference in the first place would be the higher number of cones/ganglion cells in the fovea, which naturally makes it the location from where the cortex receives maximum information. A preferential treatment would lead to further magnification in the visual pathway and the cortex.

We have completed a survey of the important pieces of literature related to the existence of a log polar transformation in the human visual system. The idea of such a transformation remains unchallenged, and researchers continue to investigate the reasons for the existence of such a transformation. We related two key explanations for such a mapping, and looked at the literature presenting the evidence for both explanations. We support one of the two explanations, and put forward our own ideas as to how such a mapping could have developed in the visual cortex.

Appendix C

Software

C2LaTeX

```
/*         This is version2.4                  */
```

This is a listing of the source code for the simulator. Included are some support functions and debugging functions which are not used in the final version.

Created by Wesley Snyder on 3/20/12. Copyright (c) 2012 North Carolina State University. All rights reserved. This program simulates area V1 of the cortex finding a straight line in an image.

```
//
```

SLDSimulation.c Version1.1 Created June 18, 2012
version2.0 Released June 22, 2012
Version1.2 Created June 23, 2012.
Version1.3 Created June 25, 2012.
Version 1.3: This version reverses the order of searching for the maxima in the accumulator. In this version, first, the accumulator is filtered by a relatively small (3×3) low pass filter. Then, an image is created by running findpsr *find peak to sidelobe ratio*. Then the maxima of that image are located. Version1.4 Created July 5, 2012.
Version2.0 Created July 9, 2012.
Version 2.0 cleans up a few messy bits of code, speeds up a few things, and emphasizes the brightest peak in the reconstructed image.
Version 2.1 allows the command line control of:

- sigma (of the edge detector),
- number of iterations
- some other stuff

Version 2.2: Created July 18, 2012 Version 2.2 includes most of the functions into a single file (primarily to make documentation easy). It also allows increasing the Q of the Gabor filter by

optionally modifying the mu loop at the beginning of the function Accumulate. The option is implemented (at this release) by using ifdef Q

Version 2,3, Created July 22, 2012
This version finds the top three lines, and shows them in the plot.

C.1 Definitions and structures

The simulation assumes that at each point in the visual field there are p-¿numorientations) orientations covering a range totaling 180 degrees. This is in agreement with literature [51] that suggests the orientation sensitivity of higher mammals (including presumably humans) is about ten degrees. The number of distinct orientations is thus 18.
The program goes through the following steps:

1. read in the input image and initialize (function initialize)
2. pass the input image through a temporal high-pass filter (function HighPass). The result of this is that over time, if the image does not change it will fade away.
3. apply a spatial high-pass filter to the image (function myflGabor).
4. For every pixel in the resulting high-pass filtered image, calculate its contribution to an accumulator. (function Accumulate) Note that there are seven possible *modes* of accumulation. Mode 5 seems to work best, and is the default. The mode is set by using the -m switch on the command line.
5. distinctive points in the accumulator are identified and marked (function findpsr). This function was so named because it was originally designed to use the maxima of the peak-to-sidelobe ratio. However, it turns out that the psr doesn't work very well.
6. for each peak in the accumulator which has been marked, draw a line on the screen.(function Houghinvsub).

The accumulators (there are two) are floating point, and have a width of $180+2\hat{P}$, where $\hat{P}$ is a pad value set by the variable PAD, which is defined int Neuralhough.h. The accumulator is similarly padded in the vertical direction. Currently PAD is 10 pixels. The purpose of the padding is to allow a peak on occur on the exact left, right, top, or bottom of the accumulator. **BY CONVENTION**, when a function is called with accumulator coordinates as arguments, the actual coordinates will be used (rho takes on values between $-\rho$ and ρ and *theta* will be an integer between 0 and 180

```
#define MINBRIGHTNESS 50
#define LASTITERATION 30


#include <flip.h>
#include <stdio.h>
#include <math.h>
#include <sys/stat.h>

#include <NeuralHough.h>
struct clp {
    char *infilename;
    int RecordFlag;
    int mode;
    int loops;
```

```
    float stddev;
    int degreespersample;
    float wavelength;
    float aspect;
    int numpeaks;
};
```

The structure *param* is used to hold more or less global information about the state of the program, such as pointers to the images, before, during, and after high pass temporal filtering, edge detector outputs, etc.

C.2 Help Function

The function *usage* explains the argument list to the program. The function is declared static to avoid linking name conflicts.

```
/*=================================================*/

/*                   usage                   */

/*=================================================*/
static void usage(void)
{
    printf("Usage: SLDSimulation -i filename [-s] [-v var][-m x] [-l iterations] -d pixelspersa
    printf("      The file name will be saved as retina.ifs if retina.ifs changes, \n");
    printf("      the HT will be updated.\n");
    printf("      If the -s flag is present, it will indicate saving the\n");
    printf("      evoluation of the accumulator in a (rather large) file.\n");
    printf("      The -m switch selects the mode of accumulation. Default is 5\n");
    printf("      The -v switch allows the user to specify the std dev (sigma) of the Gaussian
    printf("            Default is 0.75\n");
    printf("      The -l (loops) switch allows the user to specify the number of iterations. Fa
    printf("            signal-to-noise ratio, so the system is most sensitive at iteration 3.
    printf("      the -n switch controls the number of maxima to plot. Default is 3\n");

    printf("      The -d switch controls the number of degrees/sample. For example, since the r
    printf("            180 always, degrees/sample indirectly also selects the number of orient
    printf("            one might choose degrees/sample to be 10, which would cause 18 samples
    printf("            per samples would require ten samples. Default is ten degrees/sample. U
    printf("      The -w switch controls the wavelength of the Gabor filter. Default is 2.0.\n"
    printf("      The -a switch controls the aspect ratio of the Gabor filter. Default is 0.75
    exit(-1);
}
```

C.3 Parsing the Command Line

The function *clparse* reads the command line and interprets the command arguments. It also initializes several variables

```
/* -------------------CLPARSE----------------------*/
/* command line parser                             */
static void clparse(int argc,char **argv,struct clp *stuff)
{
int i;
    void usage(void);

/* set default values in returned structure */
if(argc == 1)
{
printf("You must specify arguments to this program\n");
printf("use SLDSimulation -h for more details\n");
exit(-1);
}
stuff->infilename = "";
    stuff->RecordFlag =0;
    stuff->mode =5;
    stuff->stddev=0.75;
    stuff->loops=3;
    stuff->degreespersample = 10;
    stuff->wavelength = 2.0;
    stuff->aspect = .75;
    stuff->numpeaks = 3;

for(i = 1; i< argc;i++)
{
if (*(argv[i]) != '-')
{
printf("Command line arguments must start with a dash\n");
exit(-1);
}
switch ( *(argv[i] + 1))
{
case 'h':
{
                usage();
                exit(0);
}
break;
```

```
case 'i':
stuff->infilename = argv[i+1];
i++;
break;  /* infilename   */
            case 'm':
stuff->mode = atoi(argv[i+1]);
i++;
break;  /* mode   */
case 's':
stuff->RecordFlag = 1;
break;
            case 'l':
stuff->loops = atoi(argv[i+1]);
i++;
break;  /* loops   */
            case 'n':
stuff->numpeaks = atoi(argv[i+1]);
                if(stuff->numpeaks >6)
                {printf("Cannot draw more than 6 peaks.\n");
                    exit(-1);}
i++;
break;  /* number of peaks to be shown   */
            case 'v':
stuff->stddev = atof(argv[i+1]);
i++;
break;  /* stddev of edge detector blur   */
            case 'd':
stuff->degreespersample = atoi(argv[i+1]);
i++;
break;  /* stddev of edge detector blur   */
            case 'w':
stuff->wavelength = atof(argv[i+1]);
i++;
break;  /* Gabor wavelength   */
            case 'a':
stuff->aspect = atof(argv[i+1]);
i++;
break;  /* Gabor aspect ratio   */
default:
printf("SLDSimulation: unknown option \n");
exit(-3);
break;
}
}
}
```

C.4 Saving Accumulator to Disk

This is a function used primarily for debugging. It saves the floating point accumulator to disk as a floating point ifs image. The padding is also saved.

```
/*=========================SaveAccPadded2Disk============================*/
/*                                                                       */
/*=======================================================================*/
void SaveAccPadded2Disk(float **acc,char *filename,struct param *p)
{
    IFSIMG hout;
    int hr,hc;
    float value;
    int hlen[3];

    void   copyandconvert(float **,IFSIMG);

    hlen[0]=2;hlen[1]=180+2*PAD;hlen[2]=2*p->Accrows+2*PAD;
    hout = ifscreate("float",hlen,IFS_CR_ALL,0);

    for(hr=0;hr < 2*p->Accrows+2*PAD;hr++)for(hc=0;hc < 180+PAD;hc++)
    {
        value = acc[hr][hc];
//        if(acc[hr][hc] > 10.0)printf("writing %f at %d %d to padded acc file \n",value,hr,hc)

        ifsfpp(hout,hr,hc,value);
        if(isnan(ifsfgp(hout,hr,hc)))
            printf("SaveAccPadded2Disk:nan at %d %d\n",hr,hc);fflush   (stdout);
    }
    ifspot(hout,filename);
    ifsfree(hout,IFS_FR_ALL);
}
```

C.5 Computing a histogram of the Accumulator values

This is also primarily used for debugging. It prints out the count of how many cells in the accumulator have distinctive brightness as a function of angle.

```
/*================================HistogramAccumulator==============*/

void  HistogramAccumulator(float **Acc,struct param *p)
{
    int r,c;
    float sum;
    for(c=0;c<180 + 2 * PAD;c++)
    {
        sum=0;
        for(r=0;r < 2*p->Accrows + 2 * PAD;r++)
            sum += Acc[r][c];
//        if(sum != 744.0) printf("iteration %d, %d %f\n",p->iteration,c,sum);
    }

}
```

C.6 Main Program

The program uses a 3D data set to store the output of the orientation-sensitive edge detectors. The name of that image is Gaborout.ifs. The main program initializes all the data structures and runs the program.

The input image is read from a disk into memory, and then high-pass filtered with respect to time. In this way, if the same stimulus is presented only once, it will fade over time. Each time through the main loop, the program checks the creation date on the input file (retina.ifs). If the file has changed, it will be read and input to the filter.

```
/*=====================================================================*/
/*                                                                     */
/*                              main                                   */
/*                                                                     */
/*=====================================================================*/

int main(int argc, char *argv[])
{


    int displayindex;       // number required to know which display to use
    struct param mparam,*p; // a list of global parameters
    p=&mparam;

    int inititalize(char *, struct param *);
    int HighPassFilter(struct param *,int);
    extern void copyandconvert(float **, IFSIMG);
    int Gabors(IFSIMG ,IFSIMG, float ,float,float);
    int WriteToIFSDisplayWindow(int,IFSIMG,int,float,int,int);
    int Accumulate(IFSIMG,float **, struct param *,int);
    void writefile(IFSIMG ,char * ,int );
    void usage(void);
    int findpsr1(struct param *);
    float findpsr2(struct param *);

    void saveacc(struct param *);
     void  clparse(int ,char *[],struct clp *);



    void zap(IFSIMG);
    void zapacc(float **,struct param *);
    void clparse(int,char *[],struct clp *);
```

```
    void clip(IFSIMG);
    struct stat mystat, *info;
    int oldmtime;
    int status;
    struct clp myclp, *stuff;
```

The accumulator is 180 degrees wide, and divided into NUMORIENTATIONS blocks. This variable is set in the include file to be 18. However, in modifying the program, be careful. The numbers 180, 18, or 10 (degrees per sample) may be used without reverence to this macro. This inconsistency will be corrected in future releases.

It is important to remember that the accumulator is actually a doubly-indexed floating point array. This array has dimensions $(2PAD + 180) \times 2Accrows$. The accumulator itself is 180 columns wide, but is imbedded in a larger array to avoid boundary problems

```
    stuff=&myclp;
    clparse(argc,argv,stuff);
//    printf("SLDSimulation version 2.1\n");
    p->degreespersample =  stuff->degreespersample;
    p->numpeaks = stuff->numpeaks;
    p->numorientations = 180/stuff->degreespersample;
    info=&mystat;
    p->df = 180/stuff->degreespersample; // pixels per sample, also degrees per sample
    p->deltatheta = p->degreespersample / RAD2DEG; // space betweeen samples, measured in radia

    oldmtime=0;
```

The first argument is the name of the input file. That is stuff-¿infilename. The name is passed to the initialize function. First, we check that there are at least two arguments.

```
    if(stuff->RecordFlag) p->RecordFlag = 1;else p->RecordFlag=0;
    p->iteration =0;
    p->display1 = initialize(stuff->infilename,p);// initalize all images
```

The input will be read if it needs to be read and update y. Note that this is doing high pass filtering in time.

```
//    printf("Waiting for something to happen:\n");
    do
    {
        p->iteration++;
//        printf(" %d\n",p->iteration);fflush(stdout);
        stat(p->infilename,info);
        if (info->st_mtime > oldmtime)
        {

            /* enter here if the input file has been rewritten */
            /* read the new input image */
```

```
//            printf("Something Happened at iteration %d!!",p->iteration);
            oldmtime=info->st_mtime;
            ifsReRead(p->infilename,p->retina);
            p->iteration=1;
        }


    loop1:        status = HighPassFilter(p,p->iteration);
        if(status ==0)
        {
            p->iteration++;
            printf(".");
            goto loop1;
        }
```

function Gabors will compute the Gabor filter of the input at all the orientations

```
        Gabors(p->y,p->v,stuff->stddev,stuff->wavelength,stuff->aspect);

        zapacc(p->Acc1,p); // set the last version of the accumulator back to zero
```

Here is the call to the critical function: *Accumulate*. The function accumulate will add up all the inputs to the accumlator

```
        Accumulate(p->v,p->Acc1,p,stuff->mode);
```

the following two lines have been commented out to make the program run faster, but are very useful for debuggin.

```
//        HistogramAccumulator(p->Acc1,p);
//        SaveAccPadded2Disk(p->Acc1,"HoughAfterAccumulate.ifs",p);
```

the function *saveacc* is only called for debugging. It writes a large file. Before opening the output file, the function *saveacc* will check to see if the user wants to write the file. This desire is expressed by setting the -s switch

```
        saveacc(p);
        copyandconvert(p->Acc1,p->iAcc);
//        ifspot(p->iAcc,"iAcc.ifs");
```

here is where the accumulator is displayed on the screen, at the upper left of the screen

```
        WriteToIFSDisplayWindow(p->display1,p->iAcc,0,1.0,0,0);
        sleep(1);
//    usleep(250000); // wait 250 microseconds
```

now find the point with maximum peak-to-sidelobe ratio the function findpsr finds the peaks of an accumulator at a particular point

```
       if( findpsr1(p) > 0) showHoughinv(p);

    } while (p->iteration < stuff->loops);// end while loop
    sleep(5);
}// end main
```

C.7 saveacc

```
/*==================saveacc=========================*/
/* writes a series of accumlators to a 3D float disk file   */
/*                                                          */
void saveacc(struct param *p)
{
    double v;
    int r,c;
    float max,min,scale;
    int len[3];
    char filename[132];
    IFSIMG outimg;
    if(p->RecordFlag == 0 ){/*printf("Returnflag = 0");*/return;}

#ifdef PNGS
    len[0]=2;len[1]=180;len[2]=2*p->Accrows;
    outimg=ifscreate("u8bit",len,IFS_CR_ALL,0);
    for(r=0;r < 2*p->Accrows;r++)
        for(c=0;c < 180;c++)
        {
            v=p->Acc1[r+PAD][c+PAD];
            ifsfpp(outimg,r,c,v);
        }
    sprintf(filename,"saveacc%d.png",p->iteration);
    ifsCVpot(outimg,filename);
    ifsfree(outimg,IFS_FR_ALL);
#else
//    printf("Entering saveacc\n");fflush(stdout);
    if(p->iteration == LASTITERATION)
        {
//            printf("saveacc saving\n");fflush(stdout);
//            ifspot(p->savedacc,"savedacc.ifs");
            return;
        }
//        printf("saveacc adding a frame\n");fflush(stdout);
// find max and min in this frame
    max = -100000.0;min=100000.0;
    for(r=PAD;r < 2*p->Accrows+PAD;r++)
        for(c=PAD;c < 180+PAD;c++)
        {
            v = p->Acc1[r][c];
            if(v>max) max =v;
            if(v<min) min =v;
        }
    if(max != min) scale = 255.0 / (max - min);else scale =1.0;
```

```
 // now scale the image as it is stored.

    for(r=PAD;r < 2*p->Accrows+PAD;r++)
        for(c=PAD;c < 180+PAD;c++)
        {
            v=p->Acc1[r][c];
            ifsfpp3d(p->savedacc,p->iteration,r-PAD,c-PAD,(v-min)*scale);

//            if(r == 343 && c == 125)
 //               printf("saveacc:%d %d %d %f\n",p->iteration,r,c,v);fflush(stdout);

        }
#endif
//    printf("Leaving saveacc\n");fflush(stdout);
}
```

C.8 Find peaks in accumulator

```
/*======================findpsr1======================  */
```

this function searches the accumulator to find maxima and if */ that is over a threhold, marks it in the accumulator and erases all other points
marking is done by making the point negative

```
/*=========================================================*/
#define PSRTHRESH 1.0 // no longer used
int  findpsr1(struct param *p)
{
    float maxpsr,pssr;
    int maxpsrrow,maxpsrcol;
    int r,c,count;
    float value;

    extern float psr(float **,int,int,int,int);
    extern float psracc1(struct param *p,int r,int c );
    extern void remarkacc1(struct param *);
    extern void ShowAccNeighborhood(struct param *,float **,int, int);
    extern int localmax(struct param *,float **,int ,int);
    void   copyandconvert(float **,IFSIMG);
    count = 0;
```

first copy all of Acc1 to Acc2

```
    for(r=0; r < 2*p->Accrows+2*PAD ;r++) for(c=0; c < 180+2*PAD;c++)
        p->Acc2[r][c]=p->Acc1[r][c];
```

Now, Acc2 is the acc. Search Acc2 for maxima. When a max is found, the corresponding in Acc1 is made negative.

```
    for(r=-p->Accrows; r < p->Accrows ;r++) for(c=0; c < 180;c++)
    {
        if( p->Acc2[r+p->Accrows+PAD][c+PAD]>100.0 && localmax(p,p->Acc2,r,c) )
        {
//            if(c == 145)
//            {
//            printf("Acc2:");
//            ShowAccNeighborhood(p,p->Acc2,r,c);
//            printf("Acc1:");
//            ShowAccNeighborhood(p,p->Acc1,r,c);
//            }
            value = p->Acc2[r+PAD + p->Accrows][c+PAD];
            if(value > 10000.0 && localmax(p,p->Acc2,r,c))
```

```
            {
                p->Acc1[r+PAD + p->Accrows][c+PAD] =
                value * -1.0;

//                printf("find: marking acc1=%f at rho = %d theta = %d, (%d %d)\n",
//                      value, r,c,r+p->Accrows+PAD,c+PAD);fflush(stdout);
            count++;
            }
        }
            else
            {
                p->Acc1[r+PAD + p->Accrows][c+PAD] =
                p->Acc2[r+PAD + p->Accrows][c+PAD];
            }
        }

//    printf("findpsr1: marked %d points\n",count);fflush(stdout);
//    remarkacc1(p);

#define SHOWINPUTACC
#ifdef SHOWINPUTACC
    {
        void SaveAccPadded2Disk(float **acc,char *filename,struct param *p);
        SaveAccPadded2Disk(p->Acc2,"HoughPadded.ifs",p);
    }
#endif
    return count;
}

/*======================findpsr2=======================*/
/* not used in this version                            */
/*=====================================================*/
float findpsr2(struct param *p)
{
    float maxpsr,pssr;
    int maxpsrrow,maxpsrcol;
    int r,c;

    extern float psr(float **,int,int,int,int);
    printf("Entering findpsr2\n");fflush(stdout);

    maxpsr = -100000.0;
    for(r=PAD; r < 2*p->Accrows +PAD ;r++) for(c=PAD; c < 180+PAD;c++)
    {
        pssr=p->Acc1[r][c];
        if(pssr > maxpsr)
        {
            maxpsr = pssr;
```

```
            maxpsrrow = r;
            maxpsrcol = c;
        }
    }
    printf("Maximum Acc value of %f found at row = %d (rho = %d) theta = %d\n",
           maxpsr,
           maxpsrrow, maxpsrrow-(p->Accrows)-PAD,maxpsrcol-PAD); fflush(stdout);
    return maxpsr;
}
```

C.9 printmatrix

This is a debugging utility, used only in this program

```
static void printmatrix(double **m,int nr,int nc)
{
    int row,col;
    for(row = 1;row <= nr;row++)
    {
        for(col=1;col <=nc;col++)
            printf("%f ",m[row][col]);
        printf("\n");
    }

}
```

C.10 Initialization

The initialize function sets things up

```
/*=============================================================*/

/*                initialize                                   */

/*=============================================================*/
```

The initialize function sets up everything that is required it reads the input image file and computes an accumulator

```
int initialize(char *filename,struct param *p)
{
    IFSIMG input;
    int len[4];
    int displayindex;

    int CreateIFSDisplayWindow(IFSIMG,float,int,int);
    extern void copyandconvert(float **, IFSIMG);
    int ifsany2any(IFSIMG,IFSIMG);
    int f;
    float ** SetUpAccptr(IFSIMG);
    float **MakeAccumulator(int);
     extern void   testaccindexing(struct param *);
    float makekernel(float *kernel,int kernelradius);

//    printf("Entering Initializa\n");fflush(stdout);
```

Compute the β matrix which is used to estimate the position of the peak more accurately in the accumulator. this is accomplished as follows: Assume there are n possible orientations which can be estimated by an orientation-sensitive Gabor function. Further, assume the directions of maximum sensitivity of each of those Gabor functions are defined by direction vectors $b_1, b_2, ..., b_n$ is projected onto each of these direction vectors to produce a projection μ_i. The unknown gradient vector G is projected onto each direction vector obeying

$$\mu_i = b_i^T G$$

Collect all the direction vectors into a single matrix $B = [b_1, b_2, ..., b_n]$. Then the projection of all the points can be collected into a single vector equation

$$B^T G = \mu$$

. Which can be solved for a minimum-squared-error estimate of G by (showing matrix sizes)

$$G_{2\times 1} = (BB^T)^{-1}_{2\times 2} B_{2\times n} \mu_{n\times 1}$$

In the following code, the matrix $(BB^T)^{-1}B \equiv \beta$ is computed and will be stored in the structure p This implementation has not been recently tested. It should work.

```
    {
        double **B,**Btrans,**BBtrans,**BBtransinv;
```

dmatrix is rows, columns, in that order

```
        B=dmatrix(1,2,1,p->numorientations);
        Btrans=dmatrix(1,p->numorientations,1,2);
        BBtrans= dmatrix(1,2,1,2);
        BBtransinv=dmatrix(1,2,1,2);
        p->beta = dmatrix(1,2,1,p->numorientations);
```

There may be some confusion with indices below. I use matrices indexed 1...n, whereas in the other parts of the program, indices range from 0 ... n-1.

```
        for(f = 0; f < p->numorientations;f++)
        {
            B[1][f+1] = cos(p->deltatheta* f);
            B[2][f+1] = sin(p->deltatheta* f);

        }

#ifdef TESTB1
        B[1][1] =1;
        B[2][1] =0;
        B[1][2] =0;
        B[2][2] =1;
        B[1][3] =0;
        B[2][3] =0;
        B[1][4] =0;
        B[2][4] =0;

        B[1][5] =0;
        B[2][5] =0;

        B[1][6] =0;
        B[2][6] =0;

        B[1][7] =0;
        B[2][7] =0;

        B[1][8] =0;
        B[2][8] =0;

        B[1][9] =0;
        B[2][9] =0;
```

```
        B[1][10] =0;
        B[2][10] =0;
        B[1][11] =0;
        B[2][11] =0;

        B[1][12] =0;
        B[2][12] =0;
        B[1][13] =0;
        B[2][13] =0;
        B[1][14] =0;
        B[2][14] =0;

        B[1][15] =0;
        B[2][15] =0;
        B[1][16] =0;
        B[2][16] =0;
        B[1][17] =0;
        B[2][17] =0;
        B[1][18] =0;
        B[2][18] =0;

#endif
        transpose(B,2,p->numorientations,Btrans);
#ifdef TESTB
            printf("The matrix Btrans is\n");fflush(stdout);
        printmatrix(Btrans,p->numorientations,2);
#endif
        ifsmatmult(B,Btrans,BBtrans,2,p->numorientations,p->numorientations,2);
#ifdef TESTB
        printf("THe matrix BBtrans\n");fflush(stdout);
        printmatrix(BBtrans,2,2);
#endif

        ifsinverse(BBtrans,BBtransinv,2);
#ifdef TESTB
        printf("BBransinv is\n");fflush(stdout);
        printmatrix(BBtransinv,2,2);
#endif

        ifsmatmult(BBtransinv,B,p->beta,2,2,2,p->numorientations);
#ifdef TESTB
        printf("beta is\n");fflush(stdout);

        printmatrix(p->beta, 2,p->numorientations);
#endif

    }
```

```
    p->retina=ifspin(filename);
    p->infilename=filename;
    if(p->retina == 0)
    {
        printf("Could not read input file %s\n",filename);
        exit(-1);
    }
    if(p->retina->ifsdims != 2)
    {
        printf("\nThis program can only process two-dimensional images\n");
        printf("You can extract a 2D frame from a 3D image using ChooseFrame\n");
        exit(0);
    }
    len[0]=2;
    len[2]=p->nr = ifsdimen(p->retina,1);
    len[1]=p->nc = ifsdimen(p->retina,0);
    if(p->retina->dtype != IFST_32FLT)
    {
        int status;
        p->xnm1 = ifscreate("float",len,IFS_CR_ALL,0);
        status=ifsany2any(p->retina,p->xnm1); // create a floating version of the input
//        printf("any2any returned %d\n",status);fflush(stdout);
    }
    else
    {
        p->xnm1 = ifscreate("float",len,IFS_CR_ALL,0);
    }
//    printf("Init:0.2, len=%d %d %d\n",len[0],len[1],len[2]);fflush(stdout);
```

The output file is set up in such a way that the data will not be compressed. Thus, if it must be read multiple times, it will be fater to , it will be fast

```
    p->retina->comp=0;

    ifspot(p->retina,"retina.ifs"); // save the uncompressed retina image

    // create the rest of the images

    p->xn=ifscreate("float",len,IFS_CR_ALL,0);
    p->y=ifscreate("float",len,IFS_CR_ALL,0);
    p->iy=ifscreate("u8bit",len,IFS_CR_ALL,0);
    p->temp1=ifscreate("float",len,IFS_CR_ALL,0);
    p->temp2=ifscreate("float",len,IFS_CR_ALL,0);
    len[0]=3;

    len[3]=p->numorientations;

    // create a convolution kernel to be applied later to blur the accumulator
```

```
    {
        int eye,jay;
        int inducks;
        p->kernelradius = 1;// a radius of 1 produces a 3x3
        p->kernel = (float *)malloc((p->kernelradius *2+1) * (p->kernelradius * 2 +1) * sizeof
        makekernel(p->kernel,p->kernelradius);

        printf("the radius %d convolution kernel is :\n",p->kernelradius);
        inducks=0;
        for(eye=0;eye<1+2*p->kernelradius;eye++)
        {
            for(jay=0;jay<1+2*p->kernelradius;jay++) printf("%f ",p->kernel[inducks++]);
            printf("\n");
        }
    }
    fflush(stdout);
    p->v=ifscreate("float",len,IFS_CR_ALL,0);
```

create the accumulator

first, figure out how many rho values there are Accrows does NOT include padding

```
    p->Accrows= sqrt(len[1] * len[1] + len[2]* len[2]);
//    printf("Initializing Accrows to %d\n",p->Accrows);
    len[0]=2;
    len[1]=180;
    len[2]=p->Accrows * 2;// allocation for negative rho and padding will be done in MakeAccumu
    p->Acccols =  len[1];

    p->Acc1=MakeAccumulator(p->Accrows);// accrows does not include the doubling of vertical si
    p->Acc2=MakeAccumulator(p->Accrows);// that is added in the make
```

Construct an ifs image which is big enough to hold the entire accumulator including the padding

```
    len[0]=2;len[1]=180 + 2 * PAD; len[2] = 2 * p->Accrows + 2 * PAD;
    p->iAcc=ifscreate("u8bit",len,IFS_CR_ALL,0);
```

We will want to display the image, reconstructed from the Hough so we will create an image to store those results First, figure out how many rows. Thats easy we will use the number of rows in the accumulator. Thats more than necessary, but who cares?

```
    len[2]=p->Accrows * M_SQRT2 + 1.0;
```

Lets make the number of columns the same number

```
    len[1] = len[2];
    p->reconstructed=ifscreate("u8bit",len,IFS_CR_ALL,0);
    p->tempacc=ifscreate("float",len,IFS_CR_ALL,0);

    copyandconvert(p->Acc1,p->iAcc);
    p->display1 = CreateIFSDisplayWindow(p->iAcc,1.0,0,0); // set up display of Acc
```

The second display is used to display the reconstructed lines

```
    p->display2 = CreateIFSDisplayWindow(p->reconstructed,1.0,300,0);
```

the RecordFlag used below identifies whether or not to save the accumulator to a (rather large) diskfile

```
    if(p->RecordFlag == 1)
    {
        len[0]=3;len[1]=180;len[2]=2*p->Accrows;len[3]=LASTITERATION-1;
        p->savedacc = ifscreate("float",len,IFS_CR_ALL,0);

    }

//    p->Yimg = CreateIFSDisplayWindow(p->retina,1.0); // set up display of Acc
//    printf("Returned from create\n");fflush(stdout);
    // test accumulator
//    testaccindexing(p);
    return p->display1;
} // done with function initialize
```

C.11 Set up connections of Axons

An array of axons is modeled by a triply-indexed array of pointers to accumulator cells. Each pointer models a single axon. The three indices are row, column, and orientation. This function is not implemented at this release

```
#ifdef AXONARRAY
/*==================SetUpAxonArray=======================*/
/* create an array of pointers to the accumulator          */
SetUpAxonArray(struct param *p)
{
    float theta,dtheta,cth,sth;
    int f;

    dtheta = M_PI / (p->numorientations * p->df); // how many frames is also how many angles
    for(f=0;f < p->numorientations;f++)
    {
        theta=f*dtheta;
        cth = cos(theta); sth = sin(theta);
        for(r=p->nr;r>=0;r--)for(c=p->nc;c>=0;c--)
        {
        pointer = p->v->ifsptr +

        }
    }
}
#endif
```

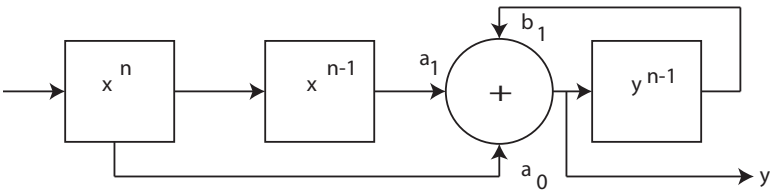


Figure C.1: The input image is high-pass filtered producing an output image y, which fades, the decay rate depends on the sampling time and the constant b_1. Here, $a_o = 1$, $a_1 = -1$, and $b_1 = .95$

C.12 High Pass Filtering

C.12.1 Dynamic Response of the input system

The dynamic response of the visual system is nicely summarized by Martinez-Conde et al. [47]: "Even our own visual system can detect stationary objects only because the image projected onto our retinas are never stationary for long." That a stationary image fades is a phenomenon referred to as "visual fading." [49, 20]. The fact that scenes we observe do not fade from view is commonly attributed to either head motion of microsaccadic motion of the eyes themselves.

In this project, we do not attempt to model microsaccadic motion, and therefore must allow constant images to fade. In the current version of the program, we also are not using camera input, but instead use a single file as input. The creation time of that file, named "img.ifs," is polled each iteration of the program, and the file is read only if it has changed. If change has occurred, it is read into an input buffer named x.

Whether read from disk or not, the input image x is subjected to a simple recursive high pass filter

$$y^n = a_o x^n - a_1 x^{n-1} + b_1 y^{n-1} \,, \tag{C.1}$$

producing output y, as illustrated in Figure C.1. The input processing, and the formation of the fading by high pass filter is performed in blocks I and II of the flow chart in Figure **??**.

In block I of that figure, the assignment $x_{n-1} \leftarrow x_n$ is denoted xnm1 :== xn, and is implemented using almost no computing resources by simply observing that these are not images but pointers to images, and swapping them in the following way:

temp = xnm1;
xnm1 = xn;
xn = temp;

```
/*=========================================================*/

/*                   HighPassFilter                        */
/*         returns 0 if the output image is null           */
/*=========================================================*/
int HighPassFilter(struct param *p,int iteration)
{
    void zap(IFSIMG);
    int ifnullimage(IFSIMG,float);
    switch(iteration)
```

```
    {

        case 1:
            zap(p->y);zap(p->xnm1);flcp(p->retina,p->xn); // initialization
            return 0;
            break;
        case 2:
            flcp(p->xn,p->y);
            flcp(p->xn,p->xnm1);
            flcp(p->retina,p->xn);
            if(ifnullimage(p->y,0)) return 0;
            break;
        default:
            flmults(p->y,p->temp1,0.9); // fade by this much
            flcp(p->temp1,p->y);
            if(ifnullimage(p->y,0)) {printf("defalt\n");return 0;}
            break;
    }
    return 1 ;
}
```

C.13 Non Maximum Suppression

function to suppress edge responses which are not maxima

```
/*=========================================================*/

/*                   nms                                   */

/*=========================================================*/
void nms(IFSIMG inimg,float angle)
{
    int r,c;
    float r1,c1,r2,c2;
    int nr,nc,ir1,ic1,ir2,ic2;
    float d,v,dmax,max;

    extern double mysin(int), mycos(int);

    nr = ifsdimen(inimg,1);nc = ifsdimen(inimg,0);
    for(r = 3;r< nr-3;r++)for(c = 3;c< nc-3;c++)
    {
        v = ifsfgp(inimg,r,c);
        if(v > 20.0)
        {
            max = 0.0;
            for(d=-2.0;d <= 3.0;d+=1.0)
            {

                r1 = r+ d* sin(angle);c1=c + d*cos(angle);
                ir1=r1;ic1=c1;
                v=ifsfgp(inimg,ir1,ic1);
                if(v > max)
                {
                    max=v;
                    dmax = d;
                }

            }
            for(d=-2.0;d <= 3.0;d+=1.0) if(d != dmax)
            {
                r2 = r+ d* sin(angle);c2=c + d*cos(angle);
                ir2=r2;ic2=c2;
//                printf("nms:suppressing %d %d because %f < %f\n",ir2,ic2,ifsfgp(inimg,r2,c2)
                ifsfpp(inimg,r2,c2,0.0);
```

```
                }
            }
        }
    }
```

C.14 Calling the Gabor Edge Detectors

The Gabor filter, applied to a point xy in an image, computes

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right)$$

Here, the Gabor is implemented by using the ifs function flGabor **The normal to the line has the angle theta, not the line itself** A 2D Gabor kernel implemented by flGabor is mathematically defined as: $G(x, y) = \exp\left[-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right] \cos\left(2\pi \frac{x'}{\lambda}\right)$ where λ is the variable *wavelength* and γ is the variable *aspect* where

$$x' = x \cos\theta + y \sin\theta \tag{C.2}$$

$$y' = -x \sin\theta + y \cos\theta \tag{C.3}$$

The parameters involved in the construction of a 2D Gabor filter are:

- The variance of the gaussian function
- The wavelength of the sinusoidal function
- The orientation of the normal to the parallel stripes of the Gabor function
- The spatial aspect ratio specifies the ellipticity of the support of the Gabor function. For , the support is circular. For the support is elongated in the orientation of the parallel stripes of the function.

Note that flGabor does not give the user the options of selecting ψ, and therefore this version of flGabor is centered around the center (since it is a cosine function. It will therefore select lines, not edges.

```
/*=========================================================*/

/*                    Gabors                               */

/*=========================================================*/
int Gabors(IFSIMG y,IFSIMG v, float stddev,float wavelength,float aspect)
{
    int i; //counter
    int len[3];
    int nf,nr,nc; //number of output frames
    int nfloats; //number of floats in a single frame
    float *oldifsptr; //save the old ifsptr
    float *fptr; // temportary pointer to a float
    double dtheta; // increment theta
    int frame;
```

```
    extern int  myflGabor(IFSIMG,IFSIMG,float,float,float,float);

    char newname[32];

    len[0]=2;

    nf = ifsdimen(v,2); // get number of frames in
    len[1]=nc=ifsdimen(v,0); len[2]=nr=ifsdimen(v,1);
    nfloats= nr*nc;
    dtheta = M_PI/nf;
    oldifsptr=(float *)v->ifsptr; //remember the old ifsptr
    fptr =oldifsptr;
    for(i=0;i<nf;i++)
    {
        float angle;
        angle = dtheta*i;
        v->ifsdims=2;
        v->ifsptr = (char  *)fptr;
```

The first argument is the input ifs image. The second argument is the 3d Gabor output image. The output image has 180/stuff-¿degreespersample frames, on for each angle. The third argument is the standard deviation of the Gaussian of the filter. the fourth argument is an angle of the gradient, in radians. An angle of 0 will be maximally sensitive to vertical edges. The final argument is not used, but is retained for compatibility with the ifs flGabor function.

```
#ifdef EDGEGABOR
        myflGabor(y,v,stddev,dtheta * i, wavelength);
#else
        flGabor(y,v,stddev,dtheta * i, wavelength,aspect);
#endif
//        flabsolute(v,v); // take magnitude of Gabor
//        nms(v,dtheta*i); // do nonmaximum suppression on the edge image
        fptr += nfloats ;

    }

    //restore the 3d nature of the image
    v->ifsptr = (unsigned char *)oldifsptr;
    v->ifsdims=3;
    ifspot(v,"Gaborout.ifs");

    // debug it
#ifdef DEBUG1
    for(frame=0;frame < p->numorientations;frame++)
    {
        float max;int rmax,cmax,r,c;
        printf("looking at frame %d\n",frame);
        max = -10000.0;
        for(r=0;r < nr;r++)for(c=0;c<nc;c++)
```

```
        {
//              printf("Reading %d %d\n",r,c);
            if(ifsfgp3d(v,frame,r,c) > max)
            {
                max = ifsfgp3d(v,frame,r,c);
                rmax = r;cmax =c;
            }
        }
        printf("Max at row %d col %d in frame %d\n",rmax,cmax,frame);
        printf("frame %d row %d col %d = %f %f %f %f %f\n",frame,rmax,cmax,
               ifsfgp3d(v,frame,rmax,cmax -2),
               ifsfgp3d(v,frame,rmax,cmax -1),
               ifsfgp3d(v,frame,rmax,cmax),
               ifsfgp3d(v,frame,rmax,cmax +1),
               ifsfgp3d(v,frame,rmax,cmax +2)
               );fflush(stdout);


    }
    exit(0);
#endif


    return 0;
}
```

C.15 Simulation of the Accumulator

The high-pass filtered image, y, provides input to an edge detection process simulating the simple cells of area V1. Each simple cell produces a number of outputs, one for each orientation sensitivity. Thus for each point in the image, a vector of responses is produced, and each position/orientation pair simulates a single axon to a single neuron in the accumulator. The image denoted as V in the flow chart is therefore a vector valued function:

$$\mathbf{v}(r, c)$$

Even though we now have a three-dimensional data structure, it is convenient to continue to call this an "image," and the elements "pixels."

In the biological accumulator, all S1 cells are simultaneously active, and therefore all cells in the accumulator may receive concurrent stimulation. In computer simulation however, we are forced to scan over **v**. The simulation process is accomplished by the following algorithm:

```
for(row=0;row< nr;row++)
for(col-0;col< nc;col++)
for(theta = 0;theta < k;theta=theta+dtheta)
{
    t1=Gabor(row,col,theta)
    p = pointerto(row,col,theta)
    d(rho,theta)=IncrementAccumulatorNeighborhood(p,t1)
    {
```

The pointers are computed in the initialization phase of the simulation. Each pointer simulates a single axon from the S1 cell at (row, col) with orientation sensitivity theta. In a biological neuron, the accumulation process occurs as a result of transfer of extracellular sodium into the cell from numerous synaptic inputs. Here we assume the presence of an interneuron which accumulates these inputs and produces the signal d, which will, in turn provide stimulus to the accumulator. This is simulated by simple addition in the function IncrementAccumulatorNeighborhood, which adds the signal t1 to the interneuron pointed to and, in a Gaussian-weighted way, to the neighborhood of that interneuron.

This process occurs in blocks III-IV of the flow chart.

```
/*=========================================================*/

/*                    Accumulate                           */

/*=========================================================*/
```

In the Accumulate function, the function is passed an argumen specifying which of several options to use for accumulation

```
int Accumulate(IFSIMG v, float ** Acc,struct param *p,int mode)
```

```
{
    double dtheta;
    double theta;
    int i,j;
    int c; // loop over columns
    int r; // loop over rows
    int f; // loop over frames (same as orientations
    int nr;// number of rows
    int nc; // number of cols
    int nf;
    int flatflag;
    int negflag;
    int irho; // integer version of rho
    float vtemp; //place to store a temporatry
    double sth, cth; // place to remember cos and sin of theta
    double rho; //the rho in the equation of a st line
    float y1,y2,y3,a,b,cee,xhat,yhat;
    int x1,x2,x3;
    float temp2;
    float *mu;
    float sum,average;
    double Gx,Gy;
    double mp1,mp2,sp1;
    int fmp1,fmp2,itheta;
    float mu180[180];
    void   FuzzAccumulator(struct param *);
    void incacc(float ,float ** , int , int,int ,struct param * ,int);
    void blurmu(float *,float *, struct param *p);
    extern void Acc2ifs(float **,char *,int);

    mu=(float *)malloc(p->numorientations * sizeof(float));
    nf = ifsdimen(v,2); // v is a 3D image
    nr = ifsdimen(v,1); // v is a 3D image
    nc = ifsdimen(v,0); // v is a 3D image
//    ifspot(p->v,"checkv.ifs");
    for(r=0;r < nr;r++)
        for(c=0;c< nc;c++)
        {
```

The following code estimates uses several different appraoches to incrementing the accumulator, given the 18 (or so) measured values at each point r, c.
First, extract all 18 measurements and find the average.

```
        sum = 0.0;
        for(f=0;f< p->numorientations;f++)
```

```
            sum += mu[f];
        average = sum / p->numorientations;
```

if the Q switch is defined, rescale the values

```
#define Q
#ifdef Q

            for(f=0;f< p->numorientations;f++)
            {
                mu[f] =  mu[f] * mu[f] / average;
            }
#endif
```

Now find the maximum and minimum

```
        mp1= 0.0;fmp1=0;mp2=0;fmp2=0;sp1= 1000000.0;
            for(f=0;f< p->numorientations;f++)
            {
                mu[f]= ifsfgp3d(v,f,r,c); // get Gabor value for this pixel
                if(mu[f] > mp1){mp1=mu[f]; fmp1 = f;} //remember biggest
                if(mu[f] < sp1) sp1=mu[f]; // also remember smallest
            }



#ifdef DEBUG
            if(r == c && mp1 != 0.0)
            {
                printf("Mu is \n");
                for(f=0;f<p->numorientations;f++) printf("%f ",mu[f]);
            }
            if (r == c&& mp1 != 0.0) printf("mp1=%f at %d\n",mp1,fmp1);
#endif

            for(f=0;f< p->numorientations;f++)
            {
                if(f != fmp1 && mu[f] > mp2){mp2=mu[f]; fmp2 = f;}
            }
//            if(r==115 && c == 115) printf("mp1 is %f",mp1);
```

now, fmp1 is the angle of the brightest and fmp2 is the second brightest

```
//            if(r == 115 && c == 115)printf("jumping to mode 5\n");
switch (mode)
    {
```

Version 0 is the classical HT. At each point, the entire accumulator is incremented. First, the maximum strength of the response is tested.

Note that theta ranges from -PAD to 180+PAD degrees. This is because the accumulator is deliberately oversized to allow smoothing.

```
case 0:

    if(mp1 > 5.0)
        for(itheta = -PAD; itheta< 180+PAD;itheta++)
    {
        incacc(mp1,Acc,r,c,itheta,p,1);
    }

    break;
```

This mode uses the p-¿numorientations measurements of directional derivitive magnitude and then linearly interpolates them over a total of 180 1-degree measurements. Those are then used to call incacc the interpolation is used to take account of the fact that we have only p-¿numorientations directional filters, not 180.

```
case 1:
{

    if(mp1 > 5.0)
    {
        blurmu(&mu[0],&mu180[0],p);

        for(i=0;i<180;i++)
            incacc(mu180[i],Acc,r,c,i,p,1);
        for(i=-PAD;i<0;i++)
            incacc(mu180[180+i],Acc,r,c,i,p,1);
        for(i=180;i<180+PAD;i++)
            incacc(mu180[180-i],Acc,r,c,i,p,1);
    }
}
    break;
```

mode 2 which uses the B matrix to make a mean-squared estimate gradient direction, using all (p-¿numorientation) measurements

```
case 2:
    {
```

Don't forget, only NUMORIENTATIONS measuremnts have been made Now multiply by the Beta matrix to get the estimate of the gradient

```
Gx=Gy=0.0;
for(i=0;i < p->numorientations;i++)
{
    Gx += p->beta[1][i+1] * mu[i];
    Gy += p->beta[2][i+1] * mu[i];

}
```

Now that we have the gradient, its angle is from the arctangent. The arctan will return numbers between -pi and pi and so we have to convert to degrees between zero and .

```
theta=atan2(Gy,Gx);
```

We are operating in only the range 0 to 180 degrees, so if we add π to an angle, we must change the sign of rho. That is accomplished by setting negflag before calling incacc.

```
if(theta < 0) {theta += M_PI; negflag = -1.0;}else negflag = 1.0;
theta *= RAD2DEG; // convert to floating version in degrees
itheta = theta+0.5; //integerize and round
```

increment acc using value from Gabor at this angle

```
//                printf("Accumulate2: found an angle of %d\n",itheta);
            incacc(mu[itheta/p->degreespersample],Acc,r,c,itheta,p,negflag);
        }//end of if statement
        break;
```

Mode 3 uses only a single direction, the direction of the maximum responding single cell

```
    case 3:
        {
            if(mp1> 5.0)
            {
                for(i=0;i < p->degreespersample;i++)
                    incacc(mp1,Acc,r,c,fmp1 * (p->degreespersample) +i,p,1.0);
            }
        }
        break;
```

Mode 4 takes the (p-¿numorientations) mu values and increments the accumultor at a point corresponging to each of by them an amount proportional to its strength.

```
    case 4:
        if(mp1>20.0)
        {
            for(i=0; i < p->numorientations;i++)
```

```
                for(j = 0;j<p->degreespersample;j++)
                {
                    itheta = i * p->degreespersample + j;
                    incacc(mu[i],Acc,r,c,itheta,p,1.0);
                }
            }
            break;
```

Mode 5 finds the brightest direction and estimates the direction as the weighted average of the max and its two nearest neighbors

```
        case 5:

            if(mp1 > 10.0 && (mp1- sp1)>0.05)
            {
```

We already know the biggest element of mu. That is at fmp with magnitude mp. So now look for the neighboring one

```
//                if(r == 115 && c == 115)
//                {
//                    printf("Accumulateafterif:m1=%f, fmp = %d\n",mp1,fmp1);fflush(stdout);
//                    exit(0);
//                }
                if(fmp1 == 0) x1 = p->numorientations-1;else x1 = fmp1-1;
                if(fmp1 == p->numorientations-1) x3 =0;else x3 = fmp1+1;
                x2=fmp1;
                y1=mu[x1];y2=mp1;y3=mu[x3];
                if((y1==y2) && (y2==y3)) flatflag = 1;
                else flatflag=0;
```

Now, we fit a parabola to these three points. Let x2=0, then x1 = -r x3=r the three quadratic equations become

$$y1 = r^2 a - rb + c \tag{C.4}$$

$$y2 = c \tag{C.5}$$

$$y3 = r^2 a + rb + c \tag{C.6}$$

if there are r degrees/sample with solution shown below

```
                if (flatflag == 0)
                {
                    cee=y2;
                    b=(y3-y1)/(2.0* p->degreespersample);
                    a=(y3 - (p->degreespersample)*b-cee)/(p->degreespersample * p->degreespersa
                    xhat = -b/(2.0 * a);
                    yhat = a * xhat * xhat +b * xhat +cee;
```

```
                    if(xhat < 0.0 && fmp1 == 0 )
                        xhat = xhat+180.0;
                    else
                        xhat = xhat + p->degreespersample* fmp1;
                itheta = xhat +0.5;
                }
                else
                {
                    itheta = p->degreespersample * x2;
                    yhat = 0.0;
                }
                incacc(yhat,Acc,r,c,itheta,p,1);
            }//end of if statement


            break;
```

case 6 finds the brightest two values and interpolates between them

```
        case 6:
            if(mp1 > 5.0)
        {
            float p1,p2,ExpectedVal;
            p1=mp1/(mp1+mp2);
            p2=mp2/(mp1+mp2);
//            if(c==196) printf("p1 = %f at theta = %d p2=%f at theta=%d\n",
//                              p1,fmp1,p2,fmp2);
            if(fmp1 == 0 && fmp2==p->numorientations-1)
                fmp2 = p->numorientations;
            if(fmp1 == p->numorientations-1 && fmp2==0)
                fmp1 = p->numorientations;
            ExpectedVal = fmp1 * p1 + fmp2 * p2;
//            itheta = ExpectedVal * 10.0;
            itheta = ExpectedVal * p->degreespersample;

//            if(r == c)
//                printf("Calling incacc with %f,%d %d %d\n",
//                       mp1,r,c,itheta);
            incacc(mp1,Acc,r,c,itheta,p,1);
        }
            break;
        default:printf("unrecognized accumulation mode\n");
            exit(-1);
    }//end switch
        }// end loops over r and c
//    Acc2ifs(p->Acc1,"Beforefuzz.ifs",2*p->Accrows);

//    FuzzAccumulator(p);// t
```

```
    free (mu);
}// end of Accumulate
```

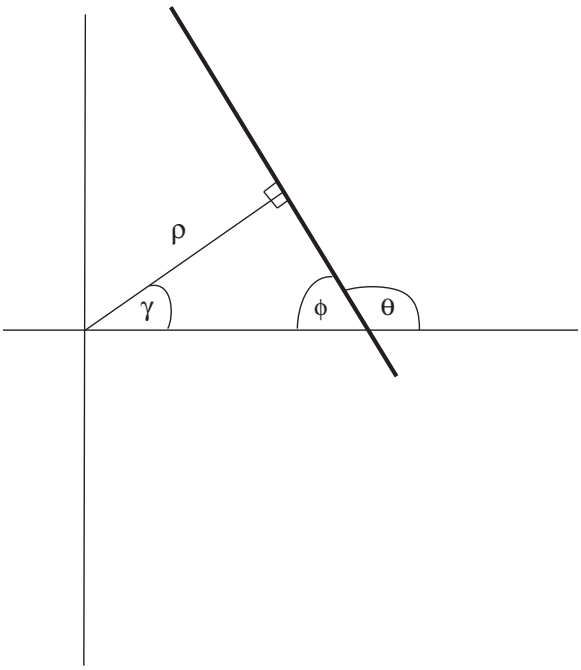


Figure C.2: Definitions of angles. The bold line is the line of interest. It sits at a normal distance ρ from the origin. The angle θ is measured by the Gabor function simulating the orientation-sensitive components of S1 cells.

C.16 incacc

This function simulates activating a single neuron by the output of a single v cell The choice of theta has already been determined. Theta measured in degrees The familiar formulation is initially used for the equation of a straight line,(see figure C.2).

$$\rho = x\cos\gamma + y\sin\gamma$$

where ρ is the distance from the origin along a normal to the line, and γ is the angle that the *normal* (not the line itself) makes with the x axis.

```
/*=========================================================================*/

/*                                incacc                                   */

/*=========================================================================*/

void incacc(float v,float ** Acc, int r, int c, int igamma,struct param *p,int negflag)
{
    int irho;
    float rho,value,accvalue;
    extern double mysin(int), mycos(int);
```

Convert γ to degrees and integerize

```
    rho = c * mycos(igamma) + r * mysin(igamma);
    rho = rho * negflag;
    irho = (rho +0.5);
    irho += p->Accrows; //since rho could be negative, shift it.
    irho += PAD;        // set the 0 0 point correctly
//    if(v >1.0 && (c == r))
//    {
//    printf("incacc:%f %d %d %d",v,r,c,igamma);
//    printf("irho = %d itheta = %d\n",irho,igamma);
//    }
        /* now irho is in array coodinates              */

    /* check valid values */
    if(irho < -PAD ){printf("neg rho %d %d\n",irho,igamma);fflush(stdout);exit(-1);}
    if(irho > 2*p->Accrows + 2 * PAD) {printf("incacc:rho too big Accrows is %d %d %d\n",p->Ac
    if (igamma < -PAD ){printf("gamma neg %d %d\n",irho,igamma);fflush(stdout);exit(-1);}
    if( igamma> (180+PAD)){printf("gamma of %d is too big  %d\n",igamma,irho);fflush(stdout);ex
//    if(c==196 && v > 26.0)
//        printf("float coordinates are %d %d %d %d\n",r,c,irho,igamma+PAD);

    accvalue = Acc[irho][igamma + PAD];
//    if(c==r) printf("incacc: updating accumulator cell %d %d %f \n",irho,(igamma+PAD),accvalu
    Acc[irho][igamma+PAD] = v+accvalue;
//    if(c==r) printf("incacc: updated accumulator cell %d %d is %f \n",irho,(igamma+PAD),Acc[
//    printf("Leaving incacc..");fflush(stdout);


}
```

C.17 Low-pass filtering the Accumulator

The function FuzzAccumulator will blur the accumulator. The result will be stored in one of the accumulator buffers, and then pointers will be swapped to avoid copying.

```
//=====================================================*/

//                 FuzzAccumulator                     */
// apply a low-pass fileter to the Acc                 */
/* result will be stored in p->Acc2, then the pointers*/
/* swapped                                             */
/* the accumulators are arrays of floats, so they have*/
/* to be turned into ifs images. Fortunately, that just*/
/* requries some pointer manipulation                  */
/*=====================================================*/

void FuzzAccumulator(struct param *p)
{
    int r,c,nr,nc,nonmax;
    int i,j;
    int inducks;
    IFSIMG hdr1,hdr2;
    float **tmpimg;
    float sum;
    int len[3];
    int passes;
    float *a1ptr,*a2ptr;
    extern void Acc2ifs(float **,char *,int);
    extern void saveacc2d(float **,struct param *,int);


//    printf("Enteringing fuzz\n");fflush(stdout);


// blur the accumulator Acc1 into Acc2 using a kxk kernel
    for(r=PAD;r < 2*p->Accrows + PAD;r++)
        for(c=PAD; c < 180 +PAD;c++)
        {
            sum =0.0;inducks=0;
            for(i=-p->kernelradius;i<=p->kernelradius;i++)
                for ( j=-p->kernelradius;j<=p->kernelradius;j++)
                {
                    sum += p->Acc1[r+i][c+j] * p->kernel[inducks++];
//                    printf("Fuzz: sum=%f\n",sum);fflush(stdout);
```

```
                }
            p->Acc2[r][c]=sum/inducks;
        }
```

accumulator 2 is the blurred accumulator. Make it acc1

```
    tmpimg = p->Acc1;p->Acc1 = p->Acc2; p->Acc1=tmpimg;
```

Now the blurred accumulator is in Acc1.

```
    Acc2ifs(p->Acc2,"BeforeBlur.ifs",2*p->Accrows);
    Acc2ifs(p->Acc1,"AfterBlur.ifs", 2*p->Accrows);
//    saveacc2d(p->Acc2,p,2);// 2 means show padding too

#ifdef NONMAX
// run nonmaximum suppression on the accumulator
    for(passes = 0;passes<2;passes++)
    for(r = PAD;r<2*(p->Accrows)+PAD;r++)
        for(c=PAD; c < 180 + PAD;c++)
        {
                {
                    nonmax=0;
                    for(i=-2;i<=2;i++)
                    {
                        for(j=-2;j<=2;j++)
                        {
                            if(p->Acc2[r][c]< p->Acc2[r+i][c+j])
                            {
                                nonmax = 1;
                                break;
                            }
                            if(nonmax) break;
                        }
                    }
                    if(nonmax) p->Acc2[r][c] = 1.0;

                }
        }


// find how many peaks there are
    for(r = PAD;r<2*(p->Accrows)+PAD;r++)
        for(c=PAD; c < 180 + PAD;c++)
        {
            if(p->Acc2[r][c] >= 3000.0)
                printf("Acc  of %f at %d (%d) %d\n",p->Acc2[r][c],r,r-p->Accrows-PAD,c);fflush
        }
#endif
    // swap the images Acc1 and Acc2
```

```
    tmpimg = p->Acc1;p->Acc1 = p->Acc2;p->Acc2=tmpimg;
//    printf("Leaving fuzz\n");fflush(stdout);

}
```

C.18 Making a Gaussian blurring kernel

```
/*                                                    */
/*====================================================*/

/*                  makekernel                        */

/*====================================================*/
/* function to make a convolution kernel              */
/*                                                    */
float makekernel(float *kernel,int kernelradius)
{
    float sigma,sigmasq,rsq,sum;
    int index;
    int i,j;
    index =0;sigma=kernelradius;sigmasq = sigma*sigma;
    sum=0;
    for(i=-kernelradius;i <=kernelradius;i++)
        for(j=-kernelradius;j <=kernelradius;j++)
        {
            rsq = i*i + j*j;
            kernel[index++]=exp(- rsq/sigmasq) / (2.0 * M_PI * sigma);
            sum += kernel[index-1];
        }
    for(index=0;index<(2*kernelradius+1) *( 2*kernelradius+1); index++)
        kernel[index]=kernel[index] / sum;
//    printf("makekernel: sums to %f\n",sum);
    return sum;
}
```

C.19 Set an image to all zeros

Because the program may run for a long time, it is necesary to sometimes set an ifs image to all zeros.

```
/*=================================================*/

/*                  zap                            */

/*=================================================*/
/* function to set an                          */
/*    ifsimage to all zeros                        */
void zap(IFSIMG z)
{
    char *ptr;
    register int i;
    int numbytes;

    numbytes=ifsdimen(z,0) * ifsdimen(z,1) * z->ifsbpd;
    ptr=z->ifsptr;
    for(i=numbytes-1;i>=0;i--)
        *ptr++ = 0.0;


}
```

C.20 Initializing an Accumulator

Since an accumulator is a pointer to a vector of pointers to floats, they cannot be zapped with the ifs zap funtion above.

```
/*=====================================================*/

/*                  zapacc                             */

/*=====================================================*/
/* function to set an accumulator                       */
/*     to all ones.                          */
void zapacc(float  **z,struct param *p)
{
    float *fptr;
    register int i;
    int numpixels;
    int nr,nc,r,c;
    numpixels= (180 + 2*PAD) * (2*p->Accrows + 2*PAD);
    fptr=&z[0][0];
    for(i=numpixels-1;i>=0;i--)
        *fptr++ = 1.0;



}
```

C.21 Disk reread

Reads an image from disk if the image has already been read once.

```
/*=================================================*/

/*                   ifsreread                     */

/*=================================================*/
/* reads a file from disk into an exisitng image     */
/* note: THIS ONLY WORKS IF THE IMAGE IS UNCOMPRESSED*/

int ifsReRead(char *filename, IFSIMG img)
{
    //     int filesize;
    //     FILE *fp, *fopen();
int nr,nc,size;
    IFSIMG in;


nr = ifsdimen(img,1); nc = ifsdimen(img,0);
// size = nr * nc * img->ifsbpd;
//    printf("reread: %d rows %d cols equals %d bytes\n",nr,nc,size);
    //     fp = fopen(filename,"rb");
    //     if(fp == NULL)
    //     {
    //         printf("Cannot read file named %s\n",filename);
    //         exit(-1);
    //     }
    //     fseek(fp,512,SEEK_CUR);
    //     fread((img+512), size, 1, fp);
    // //    memcpy((img+512),fp,size);
    // //    img->ifsptr=(char *)(img + 512);
    //     fclose(fp);
/* read the new input image */
in = ifspin(filename);
memcpy(img->ifsptr,in->ifsptr,size);

ifsfree(in,IFS_FR_ALL);

// ifspot(img,"new_img.ifs");

return 0;
}
```

C.22 Writing a file

Debuging function not currently used

```
void writefile(IFSIMG y,char * prefix,int index)
{
    char name[32];
    sprintf(name,"%s%d.ifs",prefix,index);
    printf("wriitng to the file named %s\n",name);
    ifspot(y,name);
}
```

C.23 Test for image null

```
/*                                                      */
/*            ifnullimage                               */

int ifnullimage(IFSIMG f,float value)
{
    int nr,nc,numpixels,i;
    float *ptr;
    ptr = (float *) f->ifsptr;
    nr =ifsdimen(f,1);
    nc =ifsdimen(f,0);
    numpixels = nr * nc;
//    printf("ifnullimage: %d rows, %d columns, %d floats\n",nr,nc,numpixels);
    for(i=0;i < numpixels;i++)
        if(*ptr++ != value) return 0;

    return 1;
}
```

C.24 Clip a floating point image

converts a float image to a float image with values between o and 255 Not used in this version

```
/*=================================================*/

/*                   clip                  */

/*=================================================*/


void clip(IFSIMG a)
{
    float *ptr;
    int i,n;
    n=ifsdimen(a,0) * ifsdimen(a,1);
    ptr = (float *)a->ifsptr;
    for(i=n-1;i>=0;i--)
    {
        *ptr=255.0 * 1.0
        /
        (1.0 + exp(- 0.05 * *ptr));
        ptr++;
    }
}
```

C.25 Blurring the vector of angle measurements

```
/*=================================================*/

/*                   blurmu                  */

/*=================================================*/
void blurmu(float *mu,float *mu180,struct param *p)
{
    int i,left,right;
    float mul,mur;
//    for(i=0;i<170;i++)
    for(i=0; i <(p->numorientations-1)*(p->degreespersample);i++)
    {
        left = (i/(p->degreespersample))*(p->degreespersample);
        right = (left+10);
        mul=mu[left/p->degreespersample];mur=mu[right/p->degreespersample];
//        if(mul > 100 && mur > 100)
//            printf("Blumu: l=%d r=%d i=%d mu=%f\n",left,right,i,mu[i]);fflush(stdout);
        mu180[i]=0.1 * (mul *(right - i) + mur *(i - left));
    }
    mul = mu[p->numorientations-1];mur = mu[0];
    for(i=(p->numorientations-1)*(p->degreespersample);i<180;i++)
    {
        mu180[i] = mul * (180-i) + mul * (i-((p->numorientations-1)*p->degreespersample));
//        if(mu180[i] > 10000.0)printf("blurmu:mu180[%d] = %f\n",i,mu180[i]);

    }


}
```

C.26 Display the Inverse Accumulator

```
/*====================showHoughinv==================*/
/* first, call Houghinv to get the inverse HT of the  */
/* HT we just found                                   */
/* then display it and hold awaiting a keystroke      */


int showHoughinv(struct param *p)
{
    extern Houghinvsub(float **,IFSIMG, float threshold,struct param *);
    extern VideoscaleAcc(float **,float **,struct param *);
    int nr,nc;
    int displayindex;

    int ifnullimage(IFSIMG,float);
    int WriteToIFSDisplayWindow(int,IFSIMG,int,float,int,int);


//    printf("Entering showHoughinv:\n");fflush(stdout);
#undef TESTHOUGHINV
#ifdef TESTHOUGHINV

    zapacc(p->Acc1,p);

    p->Acc1[0+p->Accrows+PAD][135+PAD] = 500;

    Houghinvsub(p->Acc1,p->reconstructed,500.0,p);


#endif
    zap(p->reconstructed); // prepare the output image

    Houghinvsub(p->Acc1,p->reconstructed,500.0,p);
    if(ifnullimage(p->reconstructed,2.0))
    {printf("reconsturcted is empty");exit(0);}
//    p->display2 = CreateIFSDisplayWindow(reconstructed,1.0);
    WriteToIFSDisplayWindow(p->display2,p->reconstructed,0,1.0,300,0);
    ifspot(p->reconstructed,"reconstructed.ifs");

}
```

C.27 MakeFake Function only for Testing

```
/*================================================*/

/*                  makefake                      */

/*================================================*/
/* creates a synthetic image to test */
void makefake(char *argv1)
{
    IFSIMG farble;
    int len[3],row;
    len[0]= 2;len[1]=256;len[2]=256;
    farble=ifscreate("float",len,IFS_CR_ALL,0);
    for(row = 50;row < 100;row++)
    {
        ifsfpp(farble,row,row,200.0); // image with a single diagonal line
        ifsfpp(farble,row,100-row,200.0); // image with a single diagonal line
    }
    ifspot(farble,argv1);
}
```

C.28 Accumulator Operations Functions

```
//
//  Accoperations.c
//
//
//  Created by Wesley Snyder on 6/13/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#define TRUE 1
#define FALSE 0
```

C.28.1 Video Scale Accumulator

```
/*============================================*/
/*            VidscaleAcc                 */
/*                                            */
/*============================================*/
/* accepts one accumulator in standared, padded*/
/* format, and video -scales it               */
VidscaleAcc(float **a, float **b,struct param *p)
{
    int r,c;
    float max,min,scale;
```

```
    max = 0.0;min=100000.0;
    for(r = PAD; r < 2*p->Accrows+PAD;r++)
        for(c = PAD; c < 180+PAD;c++)
        {
            if(a[r][c] > max) max = a[r][c];
            if(a[r][c] < min) min = a[r][c];
        }
    scale = 255.0 / ( max-min);
    for(r = PAD; r < 2*p->Accrows+PAD;r++)
        for(c = PAD; c < 180+PAD;c++)
            b[r][c] = scale*(a[r][c]-min);
}
```

C.28.2 Mark POints of Interest in Accumulator

```
/*=============================================*/
/* markacc1                                    */
/* multiply the point in the accumulator to -1 */
/* so it can be identified later               */
/* the marking is done in  second acc to avoid */
/* in-place difficulties                       */
void markacc2(struct param *p,int r,int c)
{
    p->Acc2[r+p->Accrows + PAD][c+PAD] =
    p->Acc1[r+p->Accrows + PAD][c+PAD] * -1.0;
}
void remarkacc1(struct param *p)
{
    int r,c;
    for(r=0;r < 2*p->Accrows + 2* PAD;r++)
        for(c=0;c < 180 + 2 * PAD;c++)
            if(p->Acc2[r][c] <0.0)
                p->Acc1[r][c] = p->Acc2[r][c];
}
```

C.28.3 ShowAccNeighborhood

Shows the 3×3 neighborhood of a point

```
void ShowAccNeighborhood(struct param *p,float **Acc,int r, int c)
{
    int i,j;
    printf("\n***%d %d***\n",r,c);
    for(i = -1;i <= 1 ;i++)
    {

        for(j=-1;j<=1;j++)
            printf("%f ",Acc[r + p->Accrows + PAD +i][c+PAD+j]);
```

```
        printf("\n");fflush(stdout);
    }
}
```

C.28.4 Determine if a point is a Local Maximum

```
/*=============================================*/
/* localmax                                    */
/* returns TRUE if the point of interest is a  */
/*      local maximm                           */
/* note that r and c are values in rho and theta */
/* so that r ranges from -Accrows to +accrows    */
/* this function adds the PAD                   */
int localmax(struct param *p,float **Acc,int r, int c)
{
    int nonmax;
    int i,j;
    float centervalue;
    nonmax=0;
    //
    centervalue = Acc[r+p->Accrows+PAD][c+PAD];
    /*
     for(i=r+(p->Accrows)+PAD-1;i<=r+(p->Accrows)+PAD+1;i++)
     for(j=c+PAD-1;j<=c+PAD+1;j++)if(!(i==r && j ==c))
     if(centervalue <= Acc[i][j])
     */
    for(i=-1;i<=+1;i++)
        for(j=-1;j<=+1;j++)if(!(i==0 && j ==0))
            if(centervalue <= Acc[i+r+p->Accrows +PAD][j+c+PAD])
                return FALSE;
    //    printf("localmaxacc of %f at %d %d ",centervalue,r,c);fflush(stdout);
    //    printf("which is       %f at %d %d\n",centervalue,
    //           r+p->Accrows+PAD,c+PAD);fflush(stdout);

    return TRUE;


}
```

C.28.5 Make an Accumulator

```
float ** MakeAccumulator(int rows) // accumulators have 180 columsn (+pad)
{
    float *Acc;
    float **ptr;
    float *p;
    int i;
    void * malloc(size_t);
```

```
    Acc = (float *)malloc((2*rows + 2 * PAD) * (180 + 2 * PAD) * sizeof(float));
    ptr = (float **)malloc((2*rows + 2 * PAD) * sizeof(float *));
    p=Acc;
    for(i = 0; i < 2*(rows + PAD);i++)
    {
        ptr[i] = p;
        p += (180 + 2 * PAD);
    }
    //    for(i =0;i< 180;i++)ptr[rows][i] = 128.0;
    // now the value returned by MakeAccumulator can be accessed using [][]
    //    printf("Returning from MakeAccumulatpor");fflush(stdout);
    return ptr;
}
```

C.28.6 test Indexing

This function is a debugging function.

```
/*=============testaccindexing================*/
void testaccindexing(struct param *p)
{
    int rows,columns;
    int r,c;
    float v;
    for(r = 0; r < p->Accrows;r++)
        for(c = 0; c < 180; c++)
            v=p->Acc1[r+PAD][c+PAD];
}
/*==========ReadAccumulator =============*/
/* coordinates range (in degreess) from -pad to 180 + pad*/

float  ReadAccumulator(float **A,float rho, int itheta,int rhos)
{
    int  irho;
    void exit(int);
    irho =  rho +0.5;  // round off rho
    irho += rhos + PAD;

    /* angles are also in degrees, ranging from minus PAD to 180+PAD*/

    itheta +=PAD;
    if(itheta < 0 ){printf("RD:Invalid Acc address %d\n",itheta);exit(-1);}

    return A[irho][itheta];
}
/*==========WriteAccumulator =============*/
/* coordinates range (in degreess) from -pad to 180 + pad*/
```

```
float  WriteAccumulator(float **A,float rho, float theta,int rhos,float value)
{
    int itheta, irho;
    irho =  rho +0.5;  // round off rho
    irho += rhos + PAD;

    /* angles are also in degrees, ranging from minus PAD to 180+PAD*/

    itheta = theta + PAD;


    return A[irho][itheta] = value;
}
```

C.28.7 Cosine of an Angle Specified in Degrees

```
/*=======================mycos=========================*/
double  mycos(int itheta)
/* finds the cosine and sine of an integer described in degrees*/
/* this will later be done by lookup table*/
{
    double t;
    t=itheta / RAD2DEG;
    return cos(t);
}
```

C.28.8 Since of an Angle Specified in Degrees

```
/*=======================mycos=========================*/
double  mysin(int itheta)
/* finds the cosine and sine of an integer described in degrees*/
/* this will later be done by lookup table*/
{
    double t;
    t=itheta / RAD2DEG;
    return sin(t);
}
```

C.28.9 Convert an Accumulator to an IFS Image

```
/*================================================*/
/*                                                */
/*                  Acc2ifs                       */

/*================================================*/
```

```
/* writes an accumulator out to an ifs image         */
/*   Acc2ifs(floataccumulator,filename               */
/* usually used for debugging                        */
/* display includes the pad                          */
void Acc2ifs(float **acc, char *filename,int rows)
{
    IFSIMG outimg;
    int len[3];
    int r,c;
    float *fptr;
    len[0]=2;
    len[1]=180 + 2 * PAD;
    len[2]=rows+ 2*PAD;
    outimg=ifscreate("float",len,IFS_CR_ALL,0);

    fptr=(float *)outimg->ifsptr;
    for(r = 0;r<len[2];r++)
        for(c=0;c<len[1];c++)
            *fptr++ = acc[r][c];
    ifspot(outimg,filename);
}

/*================saveacc2d===============*/
/* if padflag == 1, make an image showing */
/* the padding, if padflag == 2 dont      */
/* otherwise error message                */

void saveacc2d(float **acc,struct param *p,int padflag)
{
    IFSIMG temp;
    int len[3],r,c;
    float value;
    switch(padflag)
    {
        case 1:
            len[0]=2;len[1]=180;len[2]=2*p->Accrows;
            temp = ifscreate("float",len,IFS_CR_ALL,0);
            for(r=0;r< 2*p->Accrows;r++)
                for(c=0;c<180;c++)
                {
                    value=acc[r+PAD][c+PAD];
                    ifsfpp(temp,r,c,value);
                }
            break;
        case 2:
            len[0]=2;len[1]=180+2*PAD;len[2]=2*p->Accrows+2*PAD;
            temp = ifscreate("float",len,IFS_CR_ALL,0);
            for(r=0;r< 2*p->Accrows+2*PAD;r++)
```

```
                for(c=0;c<180+2*PAD;c++)
                {
                    value=acc[r][c];
                    ifsfpp(temp,r,c,value);
                }
            break;
        default:
            printf("Illegal pad value of %d passed to saveacc2d",
                   padflag);
            exit(-1);
    }// end switch
    ifspot(temp,"savedacc2d.ifs");
    ifsfree(temp,IFS_FR_ALL);
}
```

Utility function disc. Draw a disc on an ifs image

```
/*=================================================*/

/*                     disc                        */

/*=================================================*/
void disc(IFSIMG img,int r,int c,int rad,int brightness)
{
    float dtheta, theta,x,y,radius,frad;
    int ix,iy;
    frad = rad;
    for(radius = 2.0;radius <= frad;radius += 1.0)
    {
        dtheta = 0.5 / radius;
        for(theta = 0.0; theta < (2.0) * (M_PI);theta += dtheta)
        {
            x = c + radius * cos (theta);
            ix = x+.5;
            y = r + radius * sin (theta);
            iy = y+.5;
            ifsipp(img,iy,ix,brightness);
//            printf("r=%d c=%d theta = %f x = %d y= %d br=%d\n",r,c,theta,ix,iy,brightness);
        }
    }

}

/*=================================================*/

/*                   copyandconvert                */
```

```
/*=================================================*/
/* converts a float image to a uchar image           */
/*   copyandconvert(floatimage,ubyteimeage   */
/* used in displaying accumulator                    */


void copyandconvert(float ** img1,IFSIMG img2)
{
    unsigned char  *uptr;
    float *fptr;
    int nr,nc;
    int row,col,iv;
    float max,min;
    float vf,vi,scale;
    nr = ifsdimen(img2,1);nc = ifsdimen(img2,0);
    //    printf("CopyConvert: nr=%d nc = %d\n",nr,nc);
//    uptr=(unsigned char *)img2->ifsptr;
```

Find max and min of Accumulator

```
    max = -1000000.0;min=1000000.0;
    for(row=0;row < nr;row++)
        for(col = 0;col < nc;col++)
        {
            if(img1[row][col] > max) max = img1[row][col];
            if(img1[row][col] < min) min = img1[row][col];
        }
    scale=255.0/(max-min);
    for(row=0;row < nr;row++)
        for(col = 0;col < nc;col++)
        {
            vf=img1[row][col];
            if(isnan(vf))printf("copyandconvert nan %d %d \n",row,col);

            vi=(vf-min) *scale;
            if(vf > max / 2.0)
            {
//                printf("peaks at %d %d %f->%f\n",row,col,vf,vi);
                disc(img2,row,col,5,vi);
            }

//            *uptr++ = vi;
        }
//    printf("writing test image to foo.ifs\n");
    ifspot(img2,"foo.ifs");
```

now that all the points have been rescaled, mark the biggest with discs

```
//    uptr=(unsigned char *)img2->ifsptr;
```

```
//    for(row=0;row < nr;row++)
//        for(col = 0;col < nc;col++)
//        {
//            iv=*uptr++;
//            iv=ifsigp(img2,row,col);
//            if(iv > 128)
//            {
//                printf("iv=%d Drawing disc at %d %d\n",iv,row,col);
//                disc(img2,row,col,5,iv);
//            }
//        }

}
```

C.29 Reconstruct and Display the Original Image

```
//
//  Houghinvsub.c
//
//
//  Created by Wesley Snyder on 5/24/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//
```

Compatible with version 2.0 of SLDS

```
/* ============================Houghinvsub=============================*/
/*                                                                    */
/*   program to read an output from a hough transform and drawlines */
/*   in an image                                                      */
/*  usage: Houghinvsub(Houghimg, outimg)                                  */
/* note: by convention, Houghimg normally has 180 columns,      */
/*     (one column per degree)                                        */
/*  Parameters: d theta, the angle in degrees per column value  */
/* is usually 1, but this program will compute it, just in case */
/* the input does not have 180 columns                                */
/* it will still assume the angles range from 0 to 180 degrees  */
/* anr will be p->Accrows.                                            */
/* p->tempacc is a FLOAT, and must be vidscaled before conferting  */
/* outimg for display      */



int Houghinvsub(float **Houghin,IFSIMG outimg,float threshold,struct param *p)
{
```

```
int nr,nc;
int icase;
int len[3];
double rho,theta,dtheta,st,ct;
double x,y,max;
int iy,ix,i;
int irho,itheta;
float v,vin;
int anr,anc;
float min,mostneg;
int mostnegtheta,mostnegrho,lastrho,lasttheta;
anr= p->Accrows;
anc = 180;
dtheta =  M_PI  /  anc;
nr = ifsdimen(outimg,1);   nc = ifsdimen(outimg,0);
```

First, find the most negative point in the accumulator and remember it.

```
lastrho = 361;
for(i=0;i < p->numpeaks;i++)
{
    mostneg = 0.0;
    for (irho = -anr;irho < anr;irho++)for(itheta=0; itheta < anc;itheta++)
    {
        v=Houghin[irho+anr+PAD][itheta+PAD];
        if(v < mostneg )
        {
            mostneg = Houghin[irho+anr+PAD][itheta+PAD];
            mostnegrho = irho;
            mostnegtheta = itheta;
        }
    }
    printf("%d %d\n",mostnegrho,mostnegtheta);
    p->peak[i] = mostneg;p->ipeakrho[i]=mostnegrho;p->ipeaktheta[i] = mostnegtheta;
    Houghin[mostnegrho+anr+PAD][mostnegtheta+PAD] *= -1.0; // unmark this point
}
for(i=0;i < p->numpeaks;i++)
        Houghin[p->ipeakrho[i] + anr +PAD][p->ipeaktheta[i]+PAD] *= -1.0;

//    printf("Houghinvsub: strongest response is at rho,theta = (%d %d) = (%d %d)\n",mostne
//l now loop over the list of peaks drawing lines when needed
for (i=0;i < p->numpeaks;i++)
{
    //irho is coordinates in the accumulator
    // so irho is always positive. must convert to find rho
    irho = p->ipeakrho[i];itheta = p->ipeaktheta[i];
    if(irho == 0) rho = irho + drand48()/1000.0; else
        rho = irho;
    theta = itheta * dtheta;
```

```
        st = sin(theta); ct = cos(theta);
        vin=Houghin[irho+anr+PAD][itheta+PAD];


        if(vin <0.0)
        {
            //            printf("Houghinvsub:%d %d %f\n",irho,itheta,vin);fflush(stdout);
            if(fabs(st)<= 0.001)
            {
                for(iy=0;iy<nr;iy++)
                {
                    if(vin == mostneg)
                        ifsfpp(p->tempacc,iy,irho,-mostneg * 2.0 );
                    else
                        ifsfpp(p->tempacc,iy,irho,-vin);
                }
            }
            else if(fabs(ct) <= 0.001)
            {
                //                printf("horizontal rho=%f irho = %d, itheta=%d (theta = %f),
                for(ix=0;ix<nc;ix++)
                {
                    if(vin == mostneg)
                        ifsfpp(p->tempacc,irho,ix,-mostneg * 2.0 );
                    else
                        ifsfpp(p->tempacc,irho,ix,-vin);
                }
            }
            else
            {
#define ISINX(x) (x>=0 && x<fnc)?1:0
#define ISINY(y) (y>=0 && y<fnr)?1:0

                float x0,y0,x1,y1,alpha,length,dalpha,fnc,fnr;
                int icase;
                fnr = nr;fnc = nc;
                //                printf("\ndrawing line %d %d brightness = %f\n",irho,itheta,

                // compute where the line crosses the X axis
                icase = 0;
                if((ISINX((rho/ct))) && (ISINY((rho/st))) )        icase=icase | 1;
                else
                    if((ISINX((rho/ct))) && (ISINX((rho-nr * st)/ct))) icase=icase | 2;
                    else
                        if((ISINX((rho/ct))) && (ISINX((rho-nc * ct)/st))) icase=icase | 4;
                        else
                            if((ISINY((rho/st))) && (ISINX((rho-nr * st)/ct))) icase=icase | 8
```

```
            else
                if((ISINY((rho/st))) && (ISINY((rho-nc * ct)/st))) icase=icase
                else
                    if((ISINX(((rho- nr * st)/ct))) && (ISINY((rho-nc * ct)/st)

switch(icase)
{
    case 1:/* printf("case 1:Line crosses x=0 and y=0\n");*/
        x0=0;y0=rho/st;y1=0;x1=rho/ct;
        break;
    case 2: /*printf("case 2:Line crosses y=0 and y=nr\n");*/
        y0=0;x0=rho/ct;y1=nr;x1=(rho-nr * st)/ct;
        break;
    case 4: /*printf("case 3:Line crosses y=0 and x=nc\n");*/
        y0=0;x0=rho/ct;y1=nr;x1=(rho-nr * st)/ct;
        break;
    case 8: /*printf("case 4:Line crosses x=0 and y=nr\n");*/
        x0=0;y0=rho/st;y1=nr;x1=(rho-nr * st)/ct;
        break;
    case 16: /*printf("case 5:Line crosses x=0 and x=nc\n");*/
        x0=0;y0=rho/st;x1=nc;y1=(rho - nc * ct)/st;
        break;
    case 32: /*printf("case 6:Line crosses x=nc and y=nr\n");*/
        y0=nr;x0 = (rho - nr * st)/ct;x1=nc;y1=(rho-nc * ct)/st;
        break;
    default:printf("invalid icase value of %d ",icase);
        printf("rho=%f, st=%f ct = %f\n",rho,st,ct);
        exit(-1);
        break;

}

// compute where the line crosses the y axis



//                printf("line from x=%f y=%f to x=%f y=%f\n, brightness=%f",x0
//l the followoing code calculates all the points along a line which is
//l specified by its end points. If $x_0 and x_1, x \in \Re^n$ are the vector e
//l a line in an n-dimensional space, any point $x$ on that line between the en
//l $x=\alpha x_0 + (1- \alpha x_1$
length = sqrt((x0 - x1)*(x0-x1) + (y0-y1) * (y0-y1));
dalpha = 1.0/length;
for(alpha=dalpha;alpha < (1.0-dalpha);alpha += dalpha)
{
    x=alpha * x0 + (1.0-alpha) * x1;
    y=alpha * y0 + (1.0-alpha) * y1;
```

```
                    iy=y+0.5;ix = x+0.5;
                    if(iy<0 ) iy=0;
                    if(ix<0 ) ix=0;
                    if(iy>nr-1 ) iy=nr-1;
                    if(ix>nc-1 ) ix=nc-1;
                    if(isnan(y)) printf("Ynan");

                    if(vin == mostneg)
                        ifsfpp(p->tempacc,iy,ix,-mostneg * 2.0 );
                   else
                        ifsfpp(p->tempacc,iy,ix,-vin);
                }

            }
        }
    }
    ifsvidscale(p->tempacc,outimg,&max,&min,0);
    //    printf("Houghinvsub:vidscale returned %f %f \n",max,min);
}
```

Bibliography

[1] J. M. Allman and J. H. Kaas. The organization of the second visual area (vii) in the owl monkey: A second order transformation of the visual hemifield. *Brain Research*, 76:247 – 265, 1974.

[2] J. M. Allman and J. H. Kaas. Representation of the visual field on the medial wall of the occipital-parietal cortex in the owl monkey. *Science*, 129:572 – 575, 1976.

[3] K Arbter, W. E. Snyder, H. Burkhardt, and G. Hirzinger. Application of affine-invariant fourier descriptors to recognition of 3-d objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:640–647, 1990.

[4] P. Azzopardi and A. Cowey. Preferential representation of the fovea in the primary visual cortex. *Nature*, 361:719 – 721, 1993.

[5] P. Azzopardi and A. Cowey. The overrepresentation of the fovea and adjacent retina in the striate cortex and dorsal lateral geniculate nucleus of the macaque monkey. *Neuroscience*, 72:627 – 639, 1996.

[6] M. Balasubramanian, J. Polimeni, and E. L. Schwartz. The v1 -v2-v3 complex: quasiconformal dipole maps in primate striate and extra-striate cortex. *Neural Networks*, 15:1157 – 1163, 2002.

[7] D.H Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.

[8] J. Basak. Learning hough transform: A neural network model. *Neural Computation*, 13, 2001.

[9] J. Basak and A. Das. A neural network for learning hough transform for conoidal structures. In *International Joint Conference on Neural Networks*, 2001.

[10] J. Basak and A. Das. Hough transform network: Learning conoidal structures in a connectionist framework. *IEEE Transactions on Neural Networks*, 13(2), March 2002.

[11] J. Basak and A. Das. Hough transform network: A class of networks for identifying parametric structures. *Neurocomputing*, 51, 2003.

[12] J. Basak and S. Pal. Psycop – a psychologically motivated connectionist system for object perception. *IEEE Transactions on Neural Networks*, 6(6), November 1995.

[13] J. Basak and S. Pal. Hough transform network. *Electronics Letters*, 35(7), 1999.

[14] B.E Boser, E Sackinger, J Bromley, Y LeCun, and L Jackel. An analog neural network processor with programmable topology. *IEEE Journal of Solid-State Circuits*, 26(12), 1991.

[15] M. Boucart, F. Naili, P. Despretz, S. Defoort-Dhellemmes, and M. Fabre-Thorpe. Implicit and explicit object recognition at very large visual eccentricities: No improvement after loss of central vision. *Visual Cognition*, 18:839 – 858, 2010.

[16] Christopher Brown. Inherent bias and noise in the hough transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5, September 1983.

[17] J. Canny. A computational approach to edge detection. *PAMI*, 8(6), 1986.

[18] P.M. Daniel and D Whitteridge. The representation of the visual field on the cerebral cortex in monkeys. *J. Physiol*, 159:203–221, 1961.

[19] S.R. Deans. Hough transform from the radon transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(2), 185188, March 1981.

[20] R. Ditchburn and B. Ginsborg. Vision with a stabilized retinal image. *Nature*, 170, 1952.

[21] R. F. Dougherty, V. M. Koch, A. A. Brewer, B. Fischer, J. Modersitzki, and B. A. Wandell. Visual field representations and locations of visual areas v1/2/3 in human visual cortex. *Journal of Vision*, 3:586 – 598, 2003.

[22] B. M. Dow, A. Z. Snyder, R. G. Vautin, and R. Bauer. Magnification factor and receptive field size in foveal striate cortex of the monkey. *Exp Brain Res.*, 44:213–28, 1981.

[23] B. M. Dow, R. G. Vautin, and R. Bauer. The mapping of visual space onto foveal striate cortex in the macaque monkey. *The Journal of Neuroscience*, 5:890–902, 1985.

[24] S. A. Engel, G. H. Glover, and B. A. Wandell. Retinotopic organization in human visual cortex and the spatial precision of functional mri. *Cerebral Cortex*, 7:181 – 192, 1997.

[25] Jacob Feldman. Bayesian contour integration. *Perception and Psychophysics*, 63:1171 – 1182, 2001.

[26] D. Field, A. Hayes, and R. Hess. Contour integration by human visual system: Evidence for a local association field. *Vision Research*, 33(2), 1993.

[27] D. J. Field, A. Hayes, and R. Hess. Contour integration by the human visual system: Evidence for a local "association field". *Vision Research*, 33-2:173–193, 1993.

[28] H. J. Foley and M. W. Matlin. *Sensation and Perception*. http://www.skidmore.edu/ hfoley/Perc3.htm.

[29] W.S. Geisler and J.S. Perry. Contour statistics in natural images: Grouping across occlusions. *Visual Neuroscience*, 26:109 – 121, 2009.

[30] V. Gintautas, M. I. Ham, B. Kunsberg, S. Barr, S. P. Brumby, C. Rasmussen, J. S. George, I. Nemenman, L. M. A. Bettencourt, and G. T. Kenyon. Model cortical association fields account for the time course and dependence on target complexity of human contour perception. *PloS Computational Biology*, 2011.

[31] Brendan P. Glackin, Jim Harkin, T. Martin McGinnity, Liam P. Maguire, and Qingxiang Wu. Emulating spiking neural networks for edge detection on fpga hardware. In *FPL'09*, pages 670–673, 2009.

[32] H.P. Graf and L.D. Jackel. Analog electronic neural network circuits. *Circuits and Devices Magazine, IEEE*, 5(4):44 –49, 55, jul 1989.

[33] E. Grimson and D. Huttenlocher. On the sensitivity of the hough transform for object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3), March 1990.

[34] D.O. Hebb. *The organization of behavior*. Wiley & Sons, 1949.

[35] R. Hess and S. Dakin. Contour integration in the peripheral field. *Vision Research*, 39, 1999.

[36] P.V.C. Hough. Method and means for recognizing complex patterns. *U.S. Patent*, 3069654, 1962.

[37] D. H. Hubel. *Eye, Brain and Vision*. http://hubel.med.harvard.edu/index.html.

[38] David H. Hubel and Torsten N. Wiesel. *The Brain and Visual Perception, The Story of a 25-year Collaboration*. Oxford University Press, 2004.

[39] D.H. Hubel and T.N. Wiesel. Sequence regularity and geometry of orientation columns in the monkey striate cortex. *J.comp.Neurol.*, 158:267–293, 1974a.

[40] D.H. Hubel and T.N. Wiesel. Uniformity of monkey striate cortex:a parallel relationship between field size, scatter, and magnification factor. *J.comp.Neurol.*, 158:295–302, 1974b.

[41] Karthik Krish, Stuart Heinrich, Wesley Snyder, Halil Cakir, and Siamak Khorram. A new feature based image registration algorithm. In *ASPRS 2008 Annual Conference*, April 2008.

[42] Karthik Krish, Stuart Heinrich, Wesley Snyder, Halil Cakir, and Siamak Khorram. Global registration of overlapping images using accumulative image features. *Pattern Recognition Letters*, 31(2), January 2010.

[43] R. Linsker. From basic network principles to neural architecture (series): From basic network principles to neural architecture: Emergence of spatial-opponent cells from basic network principles to neural architecture: Emergence of orientation-selective cells from basic network principles to neural architecture: Emergence of orientation columns. In *Proceedings of the National Academy of Sciences USA*, volume 83: 7508-12, 8390-94, 8779-83, 1986.

[44] N. K. Logothetis. Vision: A window on consciousness. *Scientific American*, 281:44 – 51, 1999.

[45] W. J. Ma, V. Navalpakkam, J. M. Beck, R. van den Berg, and A. Pouget. Behavior and neural basis of near-optimal visual search. *Nature Neuroscience*, 14:783 – 790, 2011.

[46] L. P. Maguire, T. M. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin. Challenges for large-scale implementations of spiking neural networks on fpgas. *Neurocomput.*, 71:13–29, December 2007.

[47] S. Martinez-Conde, S. Macknik, and D. Hubel. The role of fixational eye movements in visual perception. *Nature Reviews: Neuroscience*, 3, March 2004.

[48] Z. Popovic and J. Sjostrand. Resolution, separation of retinal ganglion cells and cortical magnification in humans. *Vision Research*, 41:1313 – 1319, 2001.

[49] L. Riggs and F. Rattliff. The effects of counteracting the normal movements of the eye. *Journal of the Optical Society of America*, 42, 1952.

[50] M. M. Schira, A. R. Wade, and C. W. Tyler. Two-dimensional mapping of the central and parafoveal visual field to human visual cortex. *J Neurophysiol*, 97:4284 – 4295, 2007.

[51] Aniek Schoups, Rufin Vogels, Ning Qian, and Gay Orban. Practicing orientation identification improves orientation coding in v1 neurons. *Nature*, 2001.

[52] E. L. Schwartz. Spatial mapping in the primate sensory perception:analytic structure and relevance to perception. *Biol. Cybernetics*, pages 181 – 194, 1977.

[53] E. L. Schwartz. Computational anatomy and functional architecture of striate cortex : A spatial mapping approach to perceptual coding. *Vision Research*, 20:645 – 669, 1980.

[54] E. L. Schwartz, R. Desimone, T. D. Albright, and C. G. Gross. Shape recognition and inferior temporal neurons. *Proc.Natl.Acad.Sci*, 80:5776 – 5778, 1983.

[55] E.L. Schwartz. Spatial mapping in the primate sensory projection: Analytic structure and relevance to perception. *Biological Cybernetics*, 25, 1977.

[56] M. I. Sereno, A. M. Dale, J. B. Reppas, K. K. Kwong, J. W. Belliveau, T. J. Brady, B. R. Rosen, and R. B. H. Tootell. Borders of multiple visual areas in humans revealed by functional magnetic resonance imaging. *Science*, 268:889 – 893, 1995.

[57] T. Serre, M. Kouh, C. Cadieu, U. Knoblich, G. Kreiman, and T. Poggio. A theory of object recognition: computations and circuits in the feedforward path of the ventral stream in primate visual cortex. Technical Report CBCL Paper 259/AI Memo 2005-036, Massachusetts Institute of Technology, 2005.

[58] M. N. Shadlen and W. T. Newsome. Noise, neural codes and cortical organization. *Current Opinion in Neurobiology*, 4:569 – 579, 1994.

[59] M. W. Spratling. Predictive coding as a model of the v1 saliency map hypothesis. *Neural Networks*, 26, 2012.

[60] D. C. Van Essen, W. T. Newsome, and J. H. R. Mausell. The visual field representation in striate cortex of the macaque monkey: Asymmetries, anisotropies and individual variability. *Vision Research*, 24:429 – 448, 1984.

[61] H. Wassle, U. Grunert, J Rohrenbeck, and B. B. Boycott. Retinal ganglion cell density and cortical magnification factor in the primate. *Vision Research*, 30:1897 – 1911, 1990.

[62] C. F. R. Weiman. Polar exponential sensor arrays unify iconic and hough space representation. In *SPIE Conf. on Intelligent Robots and Computer Vision VIII: Algorithms and Techniques*, volume 1192, Nov. 1989.

[63] C. F. R. Weiman and G. M. Chaikin. Logarithmic spiral grids for image processing and display. *Computer Graphics and Image Processing*, 11:197 – 226, 1979.

[64] Qingxiang Wu, Martin Mcginnity, Liam Maguire, Brendan Glackin, and Ammar Belatreche. Chapter 7 learning mechanisms in networks of spiking neurons, 2007.

[65] X. Wu. An efficient antialiasing technique. *Computer Graphics*, 25(4), 1991.

[66] C. T. Zahn and R. Z. Roskies. Fourier descriptors for plane closed curves. *IEEE Transactions on Computers*, 21:269–281, 1972.

[67] L. Zhaoping and K. May. Psychophysical tests of the hypothesis of a bottom-up saliency map in primary visual cortex. *PLOS Computational Biology*, 2007.